2012-01-01

Modeling Power Consumption of High Performance Computer Clusters

Mario Caire
University of Texas at El Paso, mcaire@utep.edu

MODELING POWER CONSUMPTION OF HIGH PERFORMANCE COMPUTER CLUSTERS

MARIO ESTEBAN CAIRE

Department of Electrical and Computer Engineering

APPROVED:

David H. Williams, Ph.D., Chair

Patricia A. Nava, Ph.D.

Virgilio Gonzalez, Ph.D.

Leticia Velazquez, Ph.D.

Benjamin C. Flores, Ph.D.
Interim Dean of the Graduate School

Dedication

I dedicate this dissertation to my brother Abraham Daniel Jesus Caire

May 31, 1976 – July 2, 2010

MODELING POWER CONSUMPTION OF HIGH PERFORMANCE COMPUTER CLUSTERS

by

MARIO ESTEBAN CAIRE, M.S.
Department of Electrical and Computer Engineering

DISSERTATION

Presented to the Faculty of the Graduate School of
The University of Texas at El Paso
in Partial Fulfillment
of the Requirements
for the Degree of

DOCTOR OF PHILOSOPY

Department of Electrical and Computer Engineering
THE UNIVERSITY OF TEXAS AT EL PASO
May 2012

Acknowledgements

I would like to acknowledge many individuals who have played a role in helping me to achieve this goal. Thanks to the individuals at the Developmental English and Developmental Mathematics Department (from December 2000 to December 2006) for providing the opportunity to begin my doctoral studies at the University of Texas at El Paso. I especially thank David H. Harvey for dealing with my antics when I was a student at Cathedral High School. Thanks to the individuals at the Institute for Policy and Economic Development at UTEP as well. I especially thank Carlos Olmedo, Mathew McElroy, and Dr. Dennis Soden for providing me the opportunity to broaden my horizons and conduct survey and economic development research for the Institute. Several of the tools used in this dissertation are a direct result of research and analysis techniques I picked up at IPED (for instance factor analysis). I would also like to thank the faculty I learned from at the Electrical and Computer Engineering, Computer Science, and Mathematics Departments at UTEP. In particular I thank Dr. David H. Williams for his patience and guidance as professor, advisor, and committee chair. Last but not least, I thank my family for sticking with me throughout this period. In particular, I thank my mom for the prayers she made when I encountered roadblocks in my research.

Abstract

This dissertation describes a utilization-based power modeling approach that makes use of the /proc file system and is applied to estimating power load and energy consumption of high performance computer clusters. A full cluster power model is specified that meets the following conditions: 1) provides accurate energy estimates; 2) consists of a minimal number of variables; and 3) the selected variables provide a sufficient degree of explanation of the primary OS level contributors to power consumption. The cluster power model is based on analysis at the system or compute-node level, where various models are developed and evaluated. All models described in the current work are characterized by the set of variables that make up a *system activity matrix* (which is a sample of the utilization of a compute node from the /proc file system) and are specified by the corresponding values of the coefficients resulting from least squares minimization. System and cluster level models are evaluated using criteria that include the average of the sum of the squared residuals and the coefficient of determination. The primary assumptions in this work are 1) changes in system level utilization counters found in the /proc file system are linked to power loads captured by external Watt meters connected in series to compute nodes and 2) model coefficients derived from metered compute nodes characterize power loads non-metered compute nodes because the nodes are homogeneous. Thus, full cluster model power loads and energy consumption are obtained by applying model coefficients to cluster wide /proc based activity matrices captured for each compute node involved in a high performance computing benchmark computation. An R GUI is developed to facilitate selection of variables for the general model as well as for in-depth analysis of the links between system utilization and power consumption at the compute nodes. Results show that the /proc file system can be used to train sufficiently accurate system level models. Resulting model coefficients can be applied to activity matrices generated by arbitrary cluster nodes to estimate that node's power load and total energy consumption. This approach is different from extrapolating results from a single node to all nodes of a cluster.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

High performance computing (HPC) is increasingly constrained by power consumption and energy costs (Feng, 2003). This means the size of an HPC cluster is bounded by the amount of power that is available at existing facilities or that can be supplied to new facilities. Consider two of the primary factors in increased performance, the processor and memory. As it relates to the processor, not only have the number of transistors doubled every 18 to 24 months, power densities have doubled as well. This has been termed Moore's Law for Power Consumption, (Feng, 2003) (Hsu and Feng, 2005). As it relates to memory, projections indicate that memory power will become more dominant in future systems (Mandagere and Du, 2009) (Rambus, 2009). Thus, not only are processor and memory two primary factors in increased performance, they are also dominant factors in power consumption of HPC cluster compute nodes. Current estimates put the processor at 40% to 65% of total system power and memory consumption at approximately 10% of total system power. This means the two components alone account for up to 75% of system power.

Developing models of HPC clusters using these and other components is beneficial from a design and cost of ownership perspective. For instance, peak power is a critical factor in system design: it provides an upper bound for power consumption for granularities ranging from the system (individual compute nodes) to the full cluster, to the facility. One problem with peak power is the way it is measured. Inaccurate peak power requirements can lead to over provisioning at the facility level and inefficient use of power (Rusu et al., 2006). Thus more accurate estimates of peak power provided via power models can lead to better hardware provisioning. In addition, knowing the long term cost of a system supports management and administration. For instance, modeling power consumption based on expected workloads can be used to project long term electrical energy costs. This in turn can be used for budgeting and for determining total cost of ownership from an energy perspective.

Other reasons why HPC cluster power models are important include:

- Power measurements using power meters alone do not link component usage with total energy (Rivoire, 2008).
- Power models can be applied to predicting effects of dynamic power management policies (RSPKM-2007).
- Power consumption models provide a link between energy consumption and system utilization which helps reveal where power is used well and where it is wasted and to identify system designs that are not energy-proportional (KHLK-2009)

The research conducted for this dissertation addresses these issues by:

1) Developing a set of tools to facilitate data collection (both power meter and system utilization).
2) Applying these tools to analyze and derive a general model of power consumption for HPC clusters based on /proc file system utilization or activity counters.
3) Applying results to estimate total energy consumption of an HPC cluster.

Power analysis and modeling of a high performance computing cluster using Watt meter measurements along with utilization measurements across the entire cluster yields volumes of data. This data can be difficult to describe and analyze. Several techniques are developed in this work to facilitate large scale data collection with the aim of developing power models of entire HPC clusters. Results can be applied to estimating peak power, forecasting energy consumption of HPC workloads, and producing estimates of total cost of energy (the cost of electricity consumed to perform HPC workloads) that can be incorporated in to determining total cost of ownership.

One of the tools developed is an interactive graphical user interface implemented in the R programming language. The primary GUI is a compute node power modeling *rpanel* that simplifies and facilitates ad-hoc and systematic analysis and development of power models.

Without, this GUI, it is argued that power modeling and analysis on the scale presented in this work would take months to complete.

Various statistical tools are used to reduce the amount of data collected into those that significantly capture performance and utilization information and are strongly correlated with power consumption. The tools applied include standard deviation, correlation, the Coefficient of Determination, and factor analysis.

The combination of a GUI interface with data reduction techniques provides a powerful framework which enables power models to be developed cluster wide which are then applied to estimating total cost of ownership, from an energy perspective, of a power limited cluster.

The remainder of this dissertation is organized as follows. **Chapter 2** discusses previous work related to power modeling of individual system components, full systems and entire clusters. **Chapter 3** provides a detailed description of the type of information contained in the /proc file system and identifies which /proc files are sampled and the various counters tracked by the files. **Chapter 4** gives details on data collection. Included in the chapter are issues that arise during the data collection process including: 1) ensuring samples are not skipped (either from under or over sampling) when collecting /proc counter data; 2) efficiently sampling a large number of /proc files; 3) Watt meter data collection; and 4) a review of the HPC Challenge benchmarks. **Chapter 5** describes the power modeling and analysis methodology applied in this research. The chapter includes details on how least squares minimization is implemented in R and also discusses issues related to synchronization between Watt meter samples and /proc utilization counters. The details of three power modeling algorithms implemented in R are also provided. In **Chapter 6**, a general model for full cluster power consumption is derived. Results are analyzed and performance of various models is provided. Finally **Chapter 7** provides a summary and a short discussion on future work.

Chapter 2: Previous work

This chapter provides an overview of previous work related to power modeling of components such as a processor and memory, entire compute nodes or systems, and HPC clusters. The chapter begins by defining energy and power. This is followed by a brief overview of power modeling approaches that leads to a discussion of power measurement granularity (**Section 2.3**) and performance counters (**Section 2.4**). The latter section provides background on hardware performance counters and makes a case for applying OS level performance counters to modeling power consumption for HPC clusters. This is followed by a summary of previous work that can be applied to modeling full cluster power consumption. Various methods are described including thermal design power, nameplate ratings, processor based models, and full system based power modeling. Finally, other modeling approaches that are relevant to the context of this work are discussed.

2.1 Power and Energy

The following discussion applies specifically to electrical power and energy.

2.1.1 Power

Power, *P*, is the rate of transmission of electrical energy per unit of time measured in Watts (SPEC, 2011). Alternatively, it is the rate at which electrical energy is generated or consumed. The instantaneous power in a DC circuit is given by **Eq. 2.1** (SPEC, 2011).

Eq. 2.1 $P = V \times I$

Where

- V = Voltage measured in Volts, *V*.
- I = Current measured in Amps, *A*.

At the level of the compute node, the rate at which power is consumed or the power load of that node depends on the degree to which system components are utilized. This implies that an HPC cluster power load depends on individual system loads. Note that the efficiency of AC to DC conversion also affects total power consumed. In this work, true RMS Watts are measured at compute nodes (WU, 2008) and for simplicity, power conversion efficiencies are not considered. True power, also known as real power, a type of AC power, is given by (SPEC, 2011):

Eq. 2.2 $P_{avg} = V_{\mathrm{RMS}} \times I_{\mathrm{RMS}} \times \cos \varphi$

Where

- P_{avg} = average power
- φ = phase angle between voltage and current signals.

Sections **2.5** and **2.6** summarize various power modeling approaches that can be applied to modeling HPC compute node power loads.

2.1.2 Relationship between energy and power

Energy is the accumulated transmission rate of electrical energy (SPEC, 2011) measured in kilowatt hours (kWh). Like power, electrical energy consumed by an HPC system can be estimated in a variety of ways. One approach is to apply the relationship between energy, average power (P_{ave}), and time (T) given in **Eq. 2.3**:

Eq. 2.3 $E = P_{\mathrm{avg}} \times T$

Where T is a real time interval $T \in (t_0, t_1) = t_1 - t_0$.

Many of the studies discussed in the following sections utilize this equation to determine energy consumption. Because average power, P_{avg}, is the time average of power over time, we can rewrite **Eq. 2.3** as:

Eq. 2.4 $$E = P_{\text{avg}} \times T = \left(\frac{1}{T} \int_{t_0}^{t_1} P(t)dt \right) \times T = \int_{t_0}^{t_1} P(t)dt$$

In the equation above, $P(t)$ is a function that describes power oscillation over time (SPEC, 2011). **Eq. 2.4** states that total energy over a continuous time period (t_0, t_1) is the integral of the power function over that interval. For discrete power meter samples, this means total energy is the sum of all power samples taken within time period (t_0, t_1):

Eq. 2.5 $$E = \sum_{k=1}^{N} P(x_k)$$

Where x_k denotes the kth power sample taken at time kT_0 where $k \in \{1, \ldots, N\}$, N is an integer that denotes the total number of samples taken with $N(T_0) \leq t_1$, and T_0 is a real number that denotes the sample period, $0 < T_0 < (t_1 - t_0)$.

In this dissertation, energy estimates are calculated using **Eq. 2.5**. System load or power at sample x_k, $P(x_k)$, is modeled using least squares regression. That is, system load generated by the HPC Challenge benchmarks (discussed in **Chapter 4**) at time kT_0, is estimated using least squares coefficients applied to system activity that is sampled at a rate of one sample per second (i.e. $T_0 = 1$ second). Energy is calculated by summing across all power estimates. In the current work, total energy calculated by **Eq. 2.5** is given in kilowatt seconds, kWs, since $T_0 = 1$ second. To convert to kilowatt hours, kWh, the value resulting from **Eq. 2.5** is divided by 3,600.

2.2 Power modeling approaches

As indicated above, there are various approaches to power modeling. (Rivoire, 2008) identifies three: simulation models, analytical models, and high level black box models. The authors (Li and John, 2003) identify four software based approaches: 1) instruction level power modeling; 2) characterization-based macro-modeling; 3) cycle-accurate architectural level simulation and 4) performance counter-based power estimation. Performance counter approaches can be further categorized into hardware performance counter and operating system (OS) performance counter or event based models. From a performance point of view, the degree to which components on a system are exercised affects energy consumption (SPEC, 2011) and the degree to which components are utilized can be measured using performance counters. Thus, performance counter power modeling approaches typically measure power consumption at a suitable granularity and correlate these measurements with system activity measured by the counters. The next section discusses meter granularity with the following section providing background information on hardware performance counters. **Sections 2.5** and **2.6** then provide a summary of various power models as it relates to this work.

2.3 Power measurement granularity

HPC cluster power can be measured at various granularities, from course grained approaches such as measuring power at the facility level to fine grained methods that measure power of individual system components such as the processor, memory modules, chipset, fans, etc. There is a tradeoff when selecting a granularity. Course grained approaches simplify power measurement but lack detail. For instance, it is relatively easy to measure power at the facility level, but important details, such as which nodes are consuming the most power, cannot easily be disaggregated from such measurements. On the other hand, fine grained approaches provide a high degree of detail but tend to be intrusive and complex. For instance, if a fine grained approach involves multiple measurements for several system components (processor, memory

slots, chipsets, etc.) some form of synchronization is needed to align measurements (Bircher and John, 2007).

In this work, power is measured at the system or node level. One advantage of this approach is simplicity: a node simply has to be shut down and attached to a power meter in a daisy chain configuration. Another advantage is that power modeling at the system level enables analysis at the component and application levels and allows for extrapolating to system, rack and full HPC cluster levels (Sharma et al., 2006) (Feng and Cameron, 2006) (Fan et al., 2007).

There are also disadvantages, however. First, system level power measurements measure the aggregate power load of all components within the node. This makes it difficult to identify the primary power consumers (e.g. processor, memory, network interface card, disk, etc.). Second, for existing systems with thousands of nodes, this approach is not practical because of the cost involved in (re)equipping each node and the added complexity of collecting, synchronizing, and analyzing power meter data. This work addresses the first problem by correlating component utilization measurements collected from the /proc file system with power measurements. Scalability is addressed by sampling two nodes and modeling power consumption based on system utilization collected for an entire cluster.

2.4 Performance counter background

2.4.1 Hardware Performance Counters

Hardware performance counters track low-level events within a processor using a (limited) set of special purpose registers within the CPU (Isci and Martonosi, 2003) (Kufrin, 2005) (perfmon2, 2007) (Rivoire, 2008). Because hardware performance counters are implemented within the logic of a CPU, they are typically architecture specific or platform dependent (Kufrin, 2005) (PAPI, 2011). Hardware performance counters may also be family specific, that is, each processor in a family of processors may define different counters.

Typically, the best source for information on the available performance counters for a particular CPU is the vendor (Kufrin, 2005). This fact limits the generality of power models based on this approach. One factor that aids in generalizing hardware performance counter models is that many performance counters measure similar events. However, this requires mapping performance counters between CPU architectures (and families).

There are projects that address the portability issue. However, the primary focus of these projects is on microprocessor related performance events. One project, PAPI-C (PAPIC, 2012), provides support for non-CPU components, but current support focuses on network devices, GPUs, sensors like those tracked by lm_sensors (LM, 2012) (LM Doc, 2012), [1] and virtual machines. Other component support has to be coded using the PAPI-C Component Development Interface (PAPIC CDI, 2012).

Another complication with hardware performance counter approaches to modeling power consumption is that a limited set of counters can be tracked at one time because of the limited number of registers available on the CPU. To track more counters requires alternative approaches such as multiplexing (PAPI, 2011).

2.4.2 Operating System Performance Counters

To simplify data collection and to maximize the number of counters tracked over various components including disk and memory, this work focuses on OS level performance counters tracked by the /proc file system. Specifically, the /proc file system is the interface used to collect system performance information. This information is utilized to develop power models which are then applied to predicting load and energy consumption across an entire high performance computing cluster.

[1] Support for the Winbond W83782D hardware monitoring ASIC (Tyan, 2003) is provided by lm_sensors.

In addition to simplifying data collection and tracking various system components, the /proc file system is well suited for this research for the following reasons. First, fine sampling granularity, as is available with hardware performance counters (that is, data collection for sample periods under microsecond) is not necessary because the goal of this research is to estimate long term power consumption of HPC clusters. It is unlikely that power profiles for computational phases with durations under one second will significantly impact the long term outlook. This type of analysis is more appropriate when fine grained power profiles are the objective and when fine grained software power profiling is a goal. Second, access times for memory based counters such as those found in the /proc file system will not significantly impact performance of the modeling approach because an entire set of samples can be captured within several hundred micro-seconds or less. This sampling performance ensures that samples can be easily obtained for each sampling interval, which in this case is one second. The details of /proc file system are discussed in **Chapter 3** and sampling issues are discussed in **Chapter 4**.

2.5 Modeling HPC power

This section covers various approaches that can be used to model full cluster power consumption. Techniques range from estimating peak power using thermal design and nameplate ratings to full system power models based on performance counters, OS level information, and benchmarks.

2.5.1 Thermal Design Power

Thermal design power (TDP) is defined as the maximum power in Watts that a CPU can dissipate while running real world applications (TDP, 2011). TDP is often provided by CPU manufacturers to specify the amount of heat thermal solutions should dissipate. TDP can be used to estimate a compute node's peak power which can be extrapolated to the number of nodes in an HPC cluster. This approach has some drawbacks, however. First, TDP is typically only a fraction

of a CPU's maximum power dissipation (TDP, 2011). Second, TDP does not account for other contributors to total system power such as memory, chipset, disk, power supply inefficiency, etc. Finally, TDP is defined differently by different vendors, for instance, AMD and Intel each measure TDP using different methodologies (TDP, 2004). Thus estimating peak power for a compute node and extrapolating to a full cluster is likely to result in a significant underestimate of full cluster power with different power estimates arising from different chip vendors.

2.5.2 Nameplate Power Requirements

Nameplate power requirements are worst case power estimates of individual systems. While these ratings can be used to build course grained power consumption models, there are drawbacks:

1) While nameplate ratings tend to specify maximum operating ranges, in general, actual power consumption doesn't necessarily reach the limits specified in the ratings. In some cases, the rating is greater than actual load by at least 33% (S-WP3).
2) Attributing power to individual components is inaccurate and unreliable.

Thus while TDP under estimates HPC power, the use of nameplate ratings over estimates.

2.5.3 Power models that use hardware performance counters

Recall, hardware performance counters, also referred to as processor performance counters, are special purpose CPU registers. These registers can be configured to count micro-architectural events related to processor performance, such as cache misses, TLB misses, I/O interrupts, etc. (Isci and Martonosi, 2003) (Contreras and Martonosi, 2005) (Bircher and John, 2007) (PAPI, 2011). Hardware performance counters have been shown to be correlated with power consumption (Isci and Martonosi, 2003) (Li and John, 2003). Consequently, various

researchers have developed power modeling techniques based on hardware performance counters and these techniques can be applied to modeling full cluster power consumption. In the literature models range from single component power modeling to multi-component models that estimate a complete system. The following section describes studies that are a representative sample as it relates to component, system and cluster level power models.

2.5.4 Processor models based on hardware performance counters

As indicated, various researchers have developed CPU power models with utilization data collected from hardware performance counters. (Lewis et al., 2008) develop a linear regression model that relates hardware performance counters, thermal parameters (such as ambient temperatures and die temperatures), and processor power to run-time energy consumption. (Lungu et al., 2009) use a different approach based on correlation between a core's activity factor (*A*) and instructions per cycle (*IPC*). Activity factor describes the fraction of the circuit that is switching (Mathew, 2004). Dynamic power is modeled using a linear relationship between power and frequency:

Eq. 2.6 $$P \approx A \times f \times V^2 \approx IPC \times f \times V^2$$

Where

- A = CPU core activity factor
- f = the clock frequency
- V = voltage

Eq. 2.6 is a variation of **Eq. 2.7** (below), a model for dynamic power consumption of a CMOS circuit which describes the energy dissipated per cycle (Mathew, 2004).

Eq. 2.7 $$P_{\text{avg}} = \frac{1}{2} C \times f \times V^2$$

Where

- P_{avg} = average CPU power
- C = switched capacitance
- f = processor frequency
- V = input (supply) voltage

The primary drawback with the approach shown in **Eq. 2.6** and **2.7** is that it is specific to CMOS circuits. Thus the approach it is not suitable for other system components such as memory, disks, network interface cards, etc. Additionally, the primary hurdle with estimating activity factor using instructions per cycle (*IPC*) is that *IPC* is not a simple calculation (although hardware performance counters provide this measure). Further, *IPC* is not a metric that is tracked by the /proc file system (that is, at the time of this writing, *IPC* is not estimated by a /proc file for the cluster under study, *virgo2*).

Another approach is given in (Isci and Martonosi, 2003) where power consumption is modeled for 22 physical components of a Pentium 4 using 24 hardware performance event metrics. Component access rates, $\text{AR}(C_i)$, applied as weights, are measured directly or indirectly using one or more performance counters. Power consumption of component i, $P(C_i)$, is given by the following equation:

Eq. 2.8 $$P(C_i) = AR(C_i) \times AS(C_i) \times P_{\text{MAX}}(C_i) + P_{\text{NGCP}}(C_i)$$

Where

- C_i is the ith hardware component
- AR = Access Rate
- AS = Architectural Scaling

- P_{MAX} = maximum Power
- P_{NGCP} = Non-gated Clock Power

As with the previous studies, a drawback here is that overall power is based only on processor power consumption. In general, processor only models will typically underestimate energy consumption for full clusters since the processor accounts for 40% to 65% of a full system's energy consumption (Song et al., 2009).

2.5.5 Multi-component models

Intuitively, increasing the number of system components modeled should increase the accuracy of full system and full cluster models. (Contreras and Martonosi, 2005) describe a component level model of a host system that uses performance counters to estimate power consumption of the CPU and memory. A set of power weights map hardware performance counter values to processor and memory power consumption. This approach extends the model's generality to any type of processor that incorporates the same hardware performance counters. Equations for CPU and memory (RAM) power are given by:

Eq. 2.9 $$P_{CPU} = (\alpha_1 \times IFM) + (\alpha_2 \times DD) + (\alpha_3 \times DTLBM) + (\alpha_4 \times ITLBM) + (\alpha_5 \times IPC) + K_{CPU,idle}$$

Eq. 2.10 $$P_{MEM} = (\beta_1 \times IFM) + (\beta_2 \times DD) + K_{MEM}$$

Where

- *IFM* = Instruction Fetch Misses
- *DTLBM* = data translation lookaside buffer misses
- *ITLBM* = instruction translation lookaside buffer misses
- *IPC* = Instructions Per Cycle (*IPC*)
- *DD* = data dependency

- $K_{CPU,idle}$ = constant representing CPU idle power

Note that RAM power (a non-CPU component) is modeled using events that occur within the processor. The authors (Bircher and John, 2007) extend beyond a single component and demonstrate that performance counters can be used to model power consumption of more off-chip components. Full system power consumption is estimated using "trickle down" on-chip performance event counters applied to non-CPU power models of memory, chipset, I/O, disk and microprocessor subsystems. Six *performance events are used to model entire system using linear regression techniques*. This study is important because full system power modeling is simplified by eliminating the need for additional power sensing hardware to measure power consumption of non-processor components. This provides an alternative to developing power models using the Component Based Performance Application Programming Interface described in (PAPIC, 2012).

The advantages of the "Trickle Down" approach include:

- A methodology is provided that accounts for non-CPU power consumption which can be as much as 40 to 60 percent of total system power;
- Improves the accuracy of cluster level and HPC level power consumption by providing better estimates of an HPC nodes power load.

There are also drawbacks:

- It is hardware dependent: each processor supports different numbers and types of performance counters.
- With each processor, finding a representative set of performance counters is necessary.
- The power measurement method is highly intrusive using resistors connected in series with the power source.

- There is a power consumption lag between on-chip performance events and off-chip power consumption. This is especially a problem for components that are further away from the processor, such as the disk subsystem.

A third approach to full system modeling is given by (Carol, 2007) where desktop power consumption models are used to evaluate financial impacts on organizations of participation in distributed computing projects. The study applies regression models to quantify the *incremental increases in power consumption* over two conditions: idle and powered off. The performance counters utilized relate to CPU power states, CPU utilization, disk activity, and network activity. The power model developed is based on dual CPU and single CPU dual core Window's based systems using information obtained from perfmon.msc. This study is unique in that dual core CPU models are incorporated. A primary drawback to the method explored in this study is that the *model cannot be applied to other computers*. In order to apply the model to other systems, either 1) each component in a system should be considered and dummy variable should be assigned depending on whether or not the component is present or 2) build a model for each type of computer manufactured (Carol, 2007).

In summary, drawbacks associated with performance counter based power models include:

1) Available counters depend on manufacturer.
2) There is not a universal standard that defines trackable events (note however that (PAPI, 2011) seeks to define a standard).
3) One must first determine measures that affect power consumption and those that do not.
4) If performance counter data is collected using a low priority process, under heavy load, this process becomes starved for CPU time and samples are skipped (Carol, 2007).

5) Generalizing to other systems may not be possible unless support is provided for the same set of hardware performance counters.

2.5.6 Power models that use operating system events or counters

At the time of this writing, only one study (Heath et al., 2006) has been identified that explicitly utilizes OS level (/proc based) performance counters to model power consumption (for compute nodes and HPC clusters). The study addresses thermal management by emulating single node and cluster temperatures based on heat transfer, air flow, hardware information, and dynamic component utilization. Components considered include CPU, disk and network interface cards with utilization data obtained from the /proc file system. The authors base the amount of heat produced when performing work on average power consumed by a component. Average power is in turn defined as a function of its utilization. The following two formulas are used to model heat production Q, and power consumption, P:

Eq. 2.11 $$Q_{COMP} = P_U \times t$$

Eq. 2.12 $$P_U = P_{BASE} + (P_{MAX} - P_{BASE}) \times U$$

Where

- Q_{COMP} = the heat produced by a component which corresponds to the energy it consumes (**Eq. 2.11** is essentially **Eq. 2.5** used to estimate heat).
- P_{BASE} = the component's idle power consumption.
- P_{MAX} = the component's max power or power consumption when it is fully utilized.

In contrast to the power modeling approach applied in **Eq. 2.12**, which requires a separate formula for each component, this dissertation develops a full system model using linear

regression and multiple utilization counters which represent the system components. See **Chapter 6** for details.

Note that a few studies are worth noting as it relates to the current work. In (Wang et al., 2010), the authors use dynamic voltage and frequency scaling with rate adaptation for power-efficient CPU utilization control. While power models are not specifically developed, the authors utilize /proc/stat for data collection of CPU utilization information and monitor power consumption with a *watts up? Pro* power meter. There are studies identified by (Pathak et al., 2011) that develop power models for smartphones (Shye, 2009) (Zhang, 2009), desktops (Zeng, 2002) and servers (Fan, 2007) (Kansal, 2012) based on OS level CPU utilization. However a review of these studies does not explicitly reveal that the /proc file system is used for data collection (although one can assume so for system running an OS based on Linux).

2.6 Other related work

(Li and John, 2003) derive power models based on power consumption profiles of OS service routines. These OS routines can be thought of as fundamental units of OS execution. Under the assumption that these OS routines have predictable power profiles, they can be used to estimate application power consumption. To model power consumption of OS routines, the authors propose a simple regression where the independent variable is *IPC* and k_1 and k_0 are the regression model parameters:

Eq. 2.13 $P = k_1 \times IPC + k_0$.

(Feng et al., 2005) describe PowerPack, a toolkit for power and performance profiling of entire HPC systems that uses a direct methodology for measuring power consumption of a single node. The methodology provides direct measurement of system components. The approach is highly intrusive, requiring voltage meters attached to precision resistors inserted into the power

supply DC lines. In addition, the framework requires meter drivers, which makes it hardware dependent on the power meters. In a related study (Song et al., 2009), the authors apply the PowerPack framework to study the High Performance Computing Challenge (HPCC) benchmark suite. Power and energy profiles are developed for full clusters at both at the function and component levels.

In contrast with the latter two studies, the current work focuses on power consumption at the system level, both by measuring power consumption of an entire node and by linking power consumption to system utilization measured at the operating system level, using the /proc file system. In addition, utilization is collected for an entire set of cluster nodes using a C program that samples the /proc file system of each compute node. Techniques are also explored that disaggregate power consumption by component.

The authors (Kamil et al., 2008) make an argument for comparing systems with metrics based on sustained performance per watt. To do so necessitates establishment of power measurement methods for such systems. The authors take a benchmark level approach to power consumption by studying varying workloads and application mixes. Results support the use of High Performance Linpack (HPL) as a standard benchmark to compare power consumption of high performance computing systems. This is because HPL power consumption is very close to power consumed by compute intensive workloads. The authors then argue that a practical approach to measuring full cluster power consumption (i.e. power consumption while running HPL on an entire HPC cluster) is to measure HPL power consumption on a subset of systems, and extrapolate results to the full system. In this dissertation, the HPCC benchmarks are utilized, which include HPL as a sub-benchmark.

2.7 Chapter Summary

Power modeling based on OS level performance counters available in the /proc file system on Linux platforms offers several advantages over other approaches. The primary

advantage is that it is not necessary to instrument software or benchmarks to count instructions, functions, procedures or system calls. This means the /proc file system can easily support analysis of multiple benchmarks.

The primary advantage of the /proc file system over hardware performance counters is the inclusion of metrics that describe the whole system, versus architecture specific events. For instance, /proc provides information on network and disk utilization whereas hardware performance counters provide processor specific utilization information, such as cache misses, branch instructions, instructions issued, etc. The sampling approach presented in the current work does not require that /proc file system files be synchronized unlike the multi-component hardware performance counter approach in (PAPIC, 2012). The latter approach provides a framework for multiple components however synchronization is currently an active area of research (PAPIC CDI, 2012).

Combining /proc based performance counters with power meter measurements at the system or compute-node level provides sufficient detail to study individual phases of a benchmark computation as well modeling full cluster power loads. The work presented in this dissertation is similar to the approach presented in (Contreras and Martonosi, 2005) and (Carol, 2007). However, a full system power model is developed based on least squares regression that generalizes to any HPC compute node running Linux since the counters utilized in the current work are /proc based and in most cases chosen because they are implemented generally across Linux flavors.

Chapter 3: The /proc file system

The /proc file system is a virtual file system. Unlike binary or text files that exist on disk or other media such as compact disks or flash drives, the /proc file system consists of files and directories that reside in system memory (Grover, 2004) (Blackfin, 2012) (Salzman et al., 2005). [2] These directories and files are created dynamically "on the fly" when a user land process reads or writes a file's contents. The procfs kernel API defines the functions to create and manage /proc entries (Jones, 2006) (Mouw, 2001).

The /proc file system reflects the current state of the Linux kernel's internal data structures. **Figure 1** shows a typical directory listing. (Note: all figures and tables for **Chapter 3** are printed in **Appendix FT3**. Any figure or table appearing in this chapter is reprinted from the appendix for ease of reference.) As shown in the figure, there are two types of entries: 1) numerical directories; and 2) non-numerical directories and files. Numerical directories provide information on all currently running processes. For instance, the directory named "1" provides information related to the `init` process, the first process started when a Linux system boots (more information on `init` can be found in the `init(8)` manual page). Once a process exits, its corresponding numerical directory in the /proc file system vanishes. Non-numeric directories and files provide kernel related information and are the subject of the current work.

There are various ways to categorize /proc files. One way is to categorize them as static or dynamic. Static files like /proc/cpuinfo, /proc/modules and /proc/version, typically contain configuration information that does not change throughout the life or hardware upgrade cycle of a compute node. For instance, /proc/cpuinfo contains the following fields (among others): processor id (one per unique core on the system), processor speed in megahertz (cpu MHz), and cache size (in KB).

[2] Note that klogd logs Kernel messages. Two potential sources of log information are the /proc file system and the syscall interface (Wettstein, 1999).

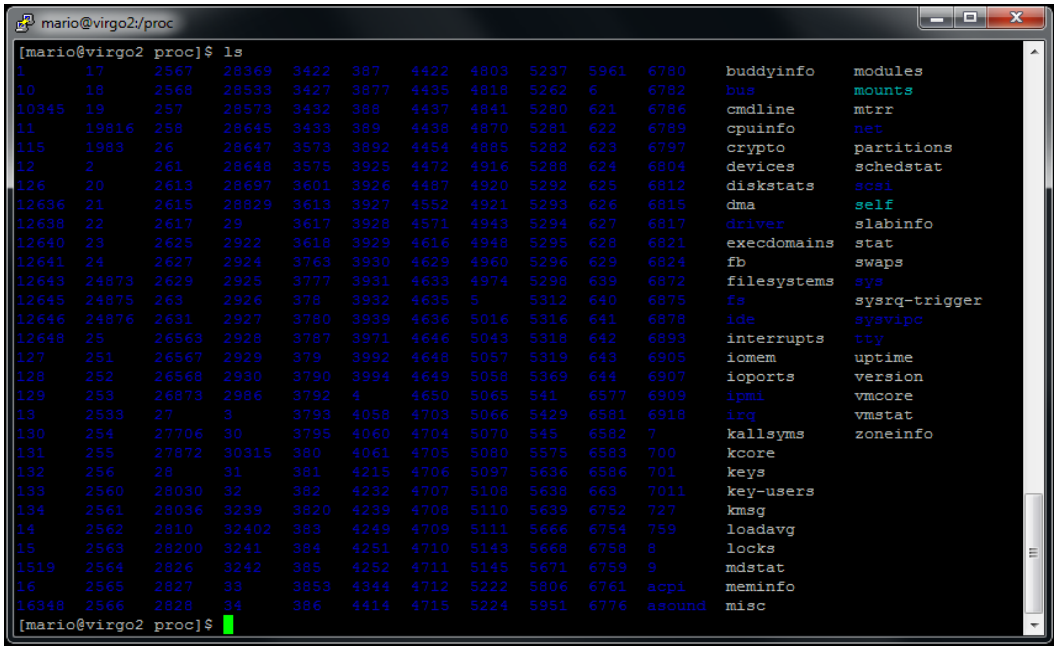


Figure 1 - Directory Listing of the /proc file system on *virgo2.ece.utep.edu*.

Dynamic files include two types: 1) those that are writeable (which allow for run-time configuration of the Linux kernel, e.g. /proc/sys/vm/drop_caches); and 2) those whose values track some form of system activity (either kernel or process). The latter types of files can include any combination of fixed metrics, cumulative counters, rates, averages, etc. These types of files are the focus of the current research because many of them provide performance information maintained by the Linux kernel that potentially can be linked to system power consumption. Delving further into their content, some files provide performance counters related to a specific system component, such as /proc/diskstats, /proc/meminfo, and /proc/net/dev, while others provide wider coverage such as /proc/interrupts and /proc/stat.

It is important to note, there is not a standard format defined for /proc file entries. Instead, each file contains a unique format that varies from a simple display of values, such as that found in /proc/loadavg (see **Figure 5**), to more complex displays of multi-row, multi-column output with row labels and column headers, for example /proc/net/dev (see **Figure 7**). **Chapter 4** provides more information on how data is extracted from /proc files with these widely ranging formats.

The power models developed in this dissertation are based on kernel specific dynamic files that track system level activity that correlate with High Performance Computing Challenge (HPCC) benchmark computations running on *virgo2.ece.utep.edu* compute nodes (see **Section 4.1** for a description of *virgo2*). The next section describes how /proc files were selected for inclusion in power model analyses with subsequent sections providing details of the selected files including the primary counters or values contained in the file. **Chapter 4** discusses the hardware specifications of *virgo2* and the HPCC benchmarks.

3.1 /proc file selection

The following table shows kernel specific /proc files and directories found on *virgo2*. Directories are listed in **bold blue** font. **Black bold** font shows files that contain system counters that change with increased computational load.

Table 1 – Non-numeric directories and files found in /proc on *virgo2.ece.utep.edu*.

acpi	execdomains	kcore	**mounts**	**sys**
buddyinfo	fb	keys	mtrr	sysrq-trigger
bus	filesystems	key-users	**net**/**dev**	**sysvipc**
cmdline	**fs**	kmsg	partitions	**tty**
cpuinfo	**ide**	**loadavg**	**schedstat**	uptime
crypto	**interrupts**	locks	**scsi**	version
devices	iomem	mdstat	**self**	vmcore
diskstats	ioports	**meminfo**	**slabinfo**	**vmstat**
dma	**irq**	misc	**stat**	**zoneinfo**
driver	kallsyms	modules	swaps	

The files in bold in **Table 1** were identified using a combination of research, analysis, and the Linux `watch` utility (more information on `watch` can be found in the `watch(1)` manual page).

The following simple procedure was applied to most of the kernel specific files found under the /proc directory: [3]

1. In a terminal window on a compute node, issue `watch –n 1 cat /proc/*path_to_proc_file*`.
2. Allow the command to run for a short time (10 to 15 seconds) while keeping track of stdout.
3. Launch an HPCC computation from the same compute node in another terminal window.
4. Watch to see if counters in the file change as the system starts computing. [4]
5. Select the *proc_file* if output from `watch` shows significant activity during the computation.

While the procedure described above is tedious and time consuming, it is fairly straight forward. Once a file has been linked to system activity using this approach, chances are this test does not have to be run again for other clusters in general. Thus the files identified in this section provide a base set of files that can be expanded as necessary based on future work.

The primary drawback with the procedure above is that it cannot be utilized to efficiently log system activity. In particular, `watch` writes to stdout (standard out, which is a terminal window by default), a resource intensive task especially if applied to a large number of compute nodes. This precludes its use for collecting sample data.

In summary, `watch` was applied in a quick-and-dirty manner to select /proc files for inclusion in power modeling analysis. If a file being analyzed showed a *significant* increase in system activity when a computation was launched, that file was selected. Of the 11 bolded files in **Table 1,** only /proc/zoneinfo was excluded from final analysis since /proc/vmstat contains the same information as /proc/zoneinfo, except summed over all zones (zoneinfo, 2011). The next

[3] Files were skipped if research indicated they did not contain performance related information or if they were purely static.

[4] One potentially useful example of `watch` described in the `watch(1)` manual page is to issue `watch uname –r`. This effectively allows an overly eager system user to watch for updates to the system kernel.

section describes each /proc file selected for analysis, file formats, and the primary variables that are show links to system activity based on the procedure described in this section.

3.2 Description of selected /proc files

Ten files were selected for final analysis. **Table 2** shows the number of variables or counters tracked per /proc file and a description of the file's main purpose. In total, 2,927 variables were logged per sample period, T_0, per compute node as described in greater detail in **Chapter 4**.

Table 2 – Number of variables tracked per /proc file.

/proc file	Counters	Purpose
buddyinfo	33	Kernel buddy memory allocation
diskstats	385	Disk I/O
interrupts	138	Interrupt usage (hardware and software)
loadavg	5	1, 5, 15 minute load averages
meminfo	30	Memory utilization and distribution
net/dev	80	Network interface counters
schedstat	672	Information on the Linux scheduler
slabinfo	1,216	Slab pool counters
stat	317	Overall system statistics
vmstat	51	Virtual memory statistics
TOTAL	**2,927**	*Total number of counters sampled per sample period*

It is important to note that not all variables reflect significant system activity. In fact, many (if not most) remain constant or nearly constant not only for the duration of an HPCC computation but also for the entire duration for which the node is turned on. Further, some of the values are static. For instance /proc/meminfo.*MemTotal* (shown in **Figure 6**) is a static value that shows the total amount of random access memory (RAM) installed on a node. Despite this, all values were tracked because of the potential to provide useful information for power modeling and analysis. In addition, tracking all proc variables at this stage provides a worst case

scenario for data collection. **Chapter 4** provides more information regarding the method used to eliminate unnecessary information from power analyses and modeling. The next ten sections discuss the details of each selected /proc file.

3.2.1 /proc/buddyinfo

The Linux kernel uses a buddy allocator to allocate chunks of memory. The /proc/buddyinfo file provides information on the buddy allocator and is useful for diagnosing memory fragmentation issues (RHEL 6, 2011) (RHEL 5, 2012) (SourceForge, 2009). **Figure 2** shows output of /proc/buddyinfo for *virgo2* compute node c0-11.

Information in this file is categorized by *node* and subcategorized by *zone* or type of memory, for instance DMA, HighMem, Normal, etc. (SourceForge, 2009). Each line in the file contains a node/zone row with each row associated with several columns of outputs (or counters). Each column position represents the number of page chunks of $2^{(Column\_Number \times PAGE\_SIZE)}$, where column numbers are labeled 0, 1, …, {*max_columns* – 1} (RHEL 6, 2011). On *virgo2* compute nodes *max_columns* = 11, which means chunks of $2^{(10 \times PAGE\_SIZE)}$. [5]

The buddyinfo file on virgo2 nodes provides *zone* information for DMA, DMA32, and Normal. Note that some kernels include DMA, Normal, and HighMEM. DMA references the first 16 MB of memory, HighMem references greater than 4 GB of memory and Normal references all memory between DMA and HighMem (RHEL 6, 2011). DMA32 is used for devices whose memory requirements are greater than 16MB but less than 4GB (Kleen, 2005). Note that when running memory intensive programs, the number of page chunks of size $2^{(0 \times PAGE\_SIZE)}$ tends to increase.

[5] According to this definition, if PAGE_SIZE = 4096 bytes and the 11th column value is "1", then a single chunk exists of $2^{(10 \times 4096)}$ bytes. Thus the size of a chunk is probably $2^{(Column\_Number)} \times$ PAGE_SIZE or "a power of 2 pages" (SourceForge, 2009). However, it is beyond the scope of the current work to verify this.

3.2.2 /proc/diskstats

This dynamic file contains disk I/O statistics for each system disk (for instance, *sda*, *fd0*, *md0*, etc.). Note that there are different files associated with disk stats between Linux 2.4 and 2.6 kernels. This section describes diskstats for Linux 2.6. **Figure 3** shows output of /proc/diskstats for *virgo2* compute node c0-11.

Table 3 – Description of /proc/diskstats *Disk* fields.

Variable	Field	Counter Description
num_r	1	# successfully completed reads
merged_r	2	# reads merged
sectors_r	3	# successfully read sectors
msec_r	4	# of milliseconds spent by all reads
complete_w	5	# successfully completed writes
merged_w	6	# writes merged
sectors_w	7	# successfully written sectors
msec_w	8	# milliseconds spent by all writes
num_io	9	# I/Os currently in progress
msec_io	10	# milliseconds spent doing I/O
msec_io_weighted	11	Weighted number of milliseconds spent doing I/O

Table 4 – Description of /proc/diskstats *Partition* fields.

Variable	Field	Counter Description
issued_r	1	# of issued reads
sectors_r	2	# of sectors requested to be read
issued_w	3	# of issued writes
sectors_w	4	# sectors requested to be written

On *virgo2*, the first 16 lines describe ram disks (labeled *ram0* through *ram15*). The following 6 lines are associated with compute node hard drives, labeled *sda*, *sda1*, *sda2*, …, *sda5*. The final three lines on each compute node are *fd0*, *hda* or *hdb*, and *md0*. It is important to note the order of the last three may vary from one node to the next, and while most compute

nodes show the device *hdb*, some show *hda*. For the purposes of this research however, the order of these rows did not pose problems because none of the counters associated with these rows proved significant for power modeling.

All lines in /proc/diskstats begin with device *major* and *minor* numbers followed by device *name*. The remaining columns vary depending on the type of device. That is, there are 11 counters for disks (for instance *ram0* through *ram15* and *sda*) and 4 counters for disk partitions (i.e. *sda0* through *sda5*). **Table 3** gives a description of output columns for disks and **Table 4** shows corresponding information for partitions. (Both tables are reprinted above). With the exception of *num_io* all counters are cumulative since boot (iostats, 2012). It is important to note, counters are 32 bit integers, and on a long lived system can wrap (iostats, 2012). To avoid problems with wrapped counters, each system was rebooted prior to benchmark testing and data collection.

3.2.3 /proc/interrupts

This file keeps track of the number of interrupts per CPU core per IO device (LPM, 2010). It shows which interrupts are in use and how many of each interrupt has occurred. Beginning with kernel 2.6.24, this file also includes interrupts internal to the Linux system that are not associated with a device. These are NMI, LOC, ERR and MIS, explained in greater detail below (LPM, 2010). **Figure 4** shows output of /proc/interrupts for *virgo2* compute node c0-11.

There are 17 to 19 rows associated with this file on *virgo2*. The first 13 to 15 identify a numeric IRQ (which is essentially column one as described below). The remaining four provide information on non-maskable interrupts (NMI), local timer interrupts (LOC), ERR and MIS. Every timer interrupt generates an NMI which is used by the NMI watchdog to detect lookups (Nguyen, 2004). LOC is the interrupt counter associated with the APIC of each CPU and ERR is incremented for errors that occur on the IO-APIC bus (Nguyen, 2004).

There are 11 columns of output. The first column refers to the IRQ number of the *interrupt in use*. Following this, there is one column for each core on the system labeled CPU0 through CPU7. Column nine, the first non-CPU column, shows the type of interrupt and the last column gives the name of the device that is located at the IRQ identified in column one (Smith, 2008). **Table 5** provides a description of each IRQ and the associated device.

In the current work, it is important to note interrupt assignments may vary between compute nodes depending on hardware available on particular nodes. For the current work, only common /proc variables are identified and selected for analysis.

Table 5 - Description of IRQs associated with /proc/interrupts on *virgo2*.

IRQ	Type	Device Name
0	IO-APIC-edge	timer
1	IO-APIC-edge	i8042
6	IO-APIC-edge	floppy
7	IO-APIC-edge	parport0
8	IO-APIC-edge	rtc
9	IO-APIC-level	acpi
12	IO-APIC-edge	i8042
14	IO-APIC-edge	ide0
58	IO-APIC-level	uhci_hcd:usb1, ehci_hcd:usb5
66	IO-APIC-level	uhci_hcd:usb3
74	IO-APIC-level	uhci_hcd:usb4, ahci (or libata)
82	IO-APIC-level	eth2
90	IO-APIC-level	HDA Intel
98	PCI-MSI	eth0
122	PCI-MSI	hda_intel
138	PCI-MSI	eth0
154	PCI-MSI	eth1
185	IO-APIC-level	uhci_hcd:usb2

3.2.4 /proc/loadavg

Figure 5 shows output of /proc/loadavg for *virgo2* compute node c0-11. This file displays five values. The first three are load average values giving number of jobs in the run queue (state R) or waiting for disk I/O (state D) averaged over 1, 5, and 15 minutes. These values provide CPU and IO utilization information and are the same values as provided by the Linux `uptime` utility (reference the `uptime(1)` manual page for more information). The remaining values are number of currently executing kernel scheduling entities (processes), number of kernel scheduling entities that currently exist on the system, and the PID of the last created process (Smith, 2008) (LPM, 2010).

3.2.5 /proc/meminfo

This file reports the amount of free and used memory (physical and swap in *k*B) including shared memory and buffers utilized by the system. **Figure 6** shows output of /proc/meminfo for *virgo2* compute node c0-11. As shown, there are 30 lines with a single value for each memory component tracked in the file. The primary fields as it relates to evaluating power models in this document are shown in **Table 6**. Note this is the same information reported by the Linux `free` utility (reference the `free(1)` manual page for more information).

Table 6 – Description of /proc/meminfo fields.

Variable	Row #	Description
memF	2	Total low and high memory that is unused
cache	4	Amount of memory in page cache
active	6	Memory that has been used recently
inactive	7	Free and available
dirty	14	Total waiting to be written to disk
writeb	15	Total actively being written to disk
pagesA	16	Non file backed pages mapped into user space page tables
slab	18	Total used by kernel to cache objects (see slablinfo)
pageTables	19	Amount of memory dedicated to lowest page table level
commitA	23	Worst case RAM estimate to complete current workload

3.2.6 /proc/net/dev

This file provides network device status information. **Figure 7** shows output of /proc/net/dev for *virgo2* compute node c0-11. The format of the file includes two header lines with an additional line for each interface. On *virgo2* nodes, interfaces include *lo0*, *eth0*, *eth1*, and *sit0*. **Table 7** lists counters tracked for each interface. The only variables considered in the current work include the number of transmitted and received packets on *lo0* and *eth0*. The latter is the primary Ethernet interface used to transmit and receive packets on the *virgo2* cluster.

Table 7 – Listing of /proc/net/dev counters.

Column	**Receive**	**Transmit**
1	Bytes	Bytes
2	Packets	Packets
3	Errs	Errs
4	Drop	Drop
5	Fifo	Fifo
6	Frame	Rolls
7	Compressed	Carrier
8	Multicast	Compressed

3.2.7 /proc/schedstat

This file shows Linux scheduler statistics for a particular compute node. **Figure 8** shows output of /proc/schedstat for *virgo2* compute node c0-11. The first entry specifies the file version. This is important because statistics provided by this file vary from one version to the next. At the time of this writing the latest version was 15 whereas the version installed on *virgo2.ece.utep.edu* was 12. The following discussion is based on version 12 of the Linux scheduler statistics file.

As noted above, the first line identifies the version. The second line contains a *timestamp* (number of seconds since UNIX epoch). The remainder of the file contains one line or row of counters for each CPU core on the system with two domain rows associated with each core. The

number of cores on a *virgo2* node is 8 (two four core processors), as indicated in /proc/cpuinfo, which in this case means there are eight CPU rows, labeled *cpu0* to *cpu7*. The corresponding domain identifiers per core are *domain0* and *domain1*.

Table 8 (reprinted below) describes CPU statistics and **Table 9** describes domain statistics. The column labeled "Variable" shows names used to identify the CPU counter, where *coreX* identifies the core number or id. Core id's can be found in /proc/cpuinfo, and in the current work range from 0 to 7.

Table 8 – Description of /proc/schedstat version 12 CPU statistics.

Variable	Field	Counter Description
cpu_*coreX*_0	1	# of times active and expired queue empty
cpu_*coreX*_1	2	# of times active queue empty
cpu_*coreX*_2	3	# of times expired queue empty
cpu_*coreX*_3	4	# of times sched_yield() called
cpu_*coreX*_4	5	**# of times the active queue has at least one other process on it**
cpu_*coreX*_5	6	# of times schedule() called
cpu_*coreX*_6	7	**# of times schedule() left processor idle**
cpu_*coreX*_7	8	# of times try_to_wake_up() called
cpu_*coreX*_8	9	# of times try_to_wake_up() process awakened last ran on waking cpu
cpu_*coreX*_9	10	**sum of all time tasks spent running on processor *X* (in ms)**
cpu_*coreX*_10	11	sum of all time tasks spent waiting to run on processor *X* (in ms)
cpu_*coreX*_11	12	# of tasks (not necessarily unique) given to processor *X*

The variables above relate to the following aspect of the Linux scheduler:

- sched_yield(), fields 1 through 4
- schedule(), fields 5 through 7
- try_to_wake_up(), fields 8 and 9
- scheduling latency, fields 10 through 12

As it relates to domain statistics contained in /proc/schedstat, the highest domain, *domain1*, "typically arbitrates balancing across all the cpus on the machine" whereas *domain0* is tightly focused "balancing only between pairs of cpus" (schedstat – Eaglet). The first field on each domain row is a bit map that identifies which cpus apply to that domain. The majority (columns 2 through 24) of domain related statistics relate to information on `load_balance()`, with the next three describing `active_load_balance()`followed by six entries that are always zero, and the remaining three providing information on `try_to_wake_up()`. Recall, **Table** 9 provides a more detailed description of domain related statistics. In the current work, *cpu_core#_4*, *cpu_core#_6*, and *cpu_core#_9* emerge as the primary variables related to both system activity and power consumption. This is discussed further in **Chapter 6**.

3.2.8 /proc/slabinfo

Kernel slab caches, or slab pools, are used to manage memory above the page level starting with Linux kernels above version 2.2 (Smith, 2008). Commonly used objects have their own slab pools (i.e. network buffers, directory cache, etc.). A slab is defined as the smallest unit that a cache can grow or shrink by and is usually a multiple of page size (Blakey, 2006). The slab cache contains a list, or cache_chain, of kmem_cache structure references (Jones, 2007) where each cache is created using a call to kmem_cache_create (Jones, 2007). Note that every cache represents storage for only one type of object (Blakey, 2006). **Figure 9** shows output of /proc/slabinfo for *virgo2* compute node c0-11. As shown, the file displays information about kernel slab caches.

As indicated previously in **Table 2**, /proc/slabinfo tracks the largest number of variables or counters. The first line of /proc/slabinfo is the version number (2.1 at the time of this writing). For most *virgo2* compute nodes, there are 152 slab caches, with 8 columns of output associated with each cache. **Table 10,** reprinted below, shows column descriptions. Note that columns 6 through 8 are tunable parameters and thus do not change with an increase or decrease in system

activity. For the current work, all counters are tracked regardless of whether or not they reflect system activity for simplicity in coding the sampling program and for providing a worst case scenario on the sampling and data collection process (see **Chapter 4**).

Table 10 – Description of /proc/slabinfo columns.

Column Label	Column #	Description
active_objs	1	# of active objects
num_objs	2	# of objects
objsize	3	Object size
objperslab	4	# objects per slab
pagesperslab	5	Pages per slab
limit	6	Max # of objects cached per CPU (tunable)
batchcount	7	Max # of objects transferred to per-CPU cache (tunable)
sharedfactor	8	Indicates sharing behavior for SMP systems (tunable)
active_slabs	9	# of active slabs per slab cache
num_slabs	10	# of slabs per slab cache
sharedavail	11	Amount of shared available per slab cache

3.2.9 /proc/stat

This pseudo-file reports information on CPU activity, interrupts, context switches, boot time, and other process related information (Smith, 2008) (LHT, 2011) (RHEL 6, 2011). **Figure 10** shows output generated by /proc/stat for *virgo2* compute node c0-11. The first 1 + *k* rows report CPU level activity where *k* is the number of CPUs or cores on the system (/proc/cpuinfo identifies each unique processor or core on the system). These rows measure the amount of time the CPU has spent performing some kind of work (LHT, 2011). The first row in the file (labeled *cpu*) reports CPU totals and the subsequent *k* rows (labeled *cpu*0, *cpu*1, …, *cpuk*) report subtotals for each CPU. The last six lines in the file correspond to *intr* (interrupts), *ctxt* (context switches), *btime* (boot time), *procs* (total processes), *procs_r* (total number of processes in the running state), and *procs_b* (total number of processes in the blocked state). The columns associated with

CPU rows are discussed next followed by a discussion of interrupt columns associated with the row labeled "*intr*".

CPU columns. Each CPU row contains nine fields (columns) that report the amount of time each CPU (processing core) has spent doing work related to one of the activities listed in **Table 11**, reprinted below. Note that Field 1 specifies which cpu the data belong to. Amount of time (for fields 2 through 8) is measured in "jiffies", USER_HZ, or 1/100th of a second (Smith, 2008) (LPM, 2005). The 5th field, *sysi* - system idle time, is equivalent to USER_HZ * {second value in /proc/uptime}. Fields 6 through 8 appear on Linux 2.6 or higher kernels and have the following meanings: *iow* (time spent waiting for I/O to complete); *hirq* and *sirq* (time servicing hardware and software interrupts respectively).

Table 11 – Description of CPU row fields found in /proc/stat.

Field No.	Field Name	Description
1	*cpu#*	CPU identifier
2	*user*	Time spent executing processes in user mode
3	*usern*	Time spent executing processes in user mode nice
4	*sys*	Time spent executing processes in system mode
5	*sysi*	Time spent idling
6	*iow*	Time spent waiting for I/O to complete
7	*hirq*	Time spent servicing hardware interrupts
8	*sirq*	Time spent servicing software interrupts
9	N/A	Always 0

Interrupt Columns. The *intr* row gives counts of serviced interrupts since the last time the system was booted, *btime*, where the first value (identified by *int1*) provides a count of the total number of system interrupts with the remaining entries showing counts for specific interrupts that occur. There are a total of 240 columns associated with this row on *virgo2*. The data collection procedure described in **Chapter 4** samples interrupt columns 1, 2, 14, 16, 76, 92 and 100.

3.2.10 /proc/vmstat

This file contains virtual memory statistics and its structure is identical to /proc/meminfo. That is, there is an identifier (name) for each variable tracked followed by a cumulative counter. Note that /proc/vmstat also contains most of the information that /proc/meminfo contains. However, both files are included in this analysis for comparison purposes. **Figure 11** shows output generated by /proc/vmstat for *virgo2* compute node c0-11.

As with all files described in this chapter, all variables are logged during the sampling process, however, only a subset of the variables are considered for actual power models based on the information they track and whether or not they reflect significant system activity. **Table 12**, reprinted below, shows selected /proc/vmstat target variables with **Table 13** showing the entire list of variables.

Table 12 – Description of /proc/vmstat target variables.

Variable	Row #	Description
nr_anon_pages	1	Mapped anonymous pages
nr_slab	4	# pages allocated by kernel slab allocator
nr_page_table_pages	5	# pages allocated to page tables
nr_dirty	6	# pages that are dirty
nr_writeback	7	# pages that are under write back
numa_hit	10	Memory allocated in intended node
numa_miss	11	Memory allocated in non-intended node
pgpgin	16	# page ins (disk reads) since last boot
pgpgout	17	# page outs (disk writes) since last boot
pgalloc_dma	20	# pages allocated to dma zone
pgalloc_dma32	21	# pages allocated to dma32 zone
pgalloc_normal	22	# pages allocated to normal zone
pgalloc_high	23	# pages allocated to highmem zone
pgfree	24	# page frees since last boot
pgfault	27	# major + minor page faults since last boot
pgmajfault	28	# major page faults since last boot

3.3 Chapter Summary

This chapter provides a detailed overview of the /proc file system. For the current work, ten files are identified and selected for analysis based on a quick-and-dirty process using the Linux `watch` utility. These files are shown in **black bold** font in **Table 2** and in **Chapter 6** are the basis of power models and analysis.

It is important to note that many /proc files are specific to a compute node's architecture (such as /proc/interrupts) and others contain different information depending on the version of the /proc file (such as /proc/schedstat). This is a similar problem as encountered with hardware performance counter approaches to power modeling: a common set of variables need to be identified to extend the generality of models developed. In the current work, system specific models are first developed and analyzed. For these analyses, variables shown in **Table 14** and **Table 15** are utilized (**Section 5.2** explains how and why these variables were selected). Based on results, a general model that can be extended to other HPC clusters is then developed using a reduced set of variables that can be applied to modeling power consumption of other Linux based HPC clusters.

Chapter 4: Data collection

The primary goals of this research are to measure power consumption on two compute nodes of a 21 node power limited Beowulf class high performance computer (HPC) cluster and to utilize kernel level /proc activity (performance utilization information) to:

- Estimate total power consumption for all nodes of the entire computing cluster.
- Determine each compute node's primary power consumers (i.e. component power consumption) in order to develop detailed power consumption profiles.
- Estimate total power and energy consumption relative to HPCC benchmark activity.

Three steps are taken to develop full cluster power consumption models: 1) *consistently* sample compute node power loads; 2) *consistently* sample /proc file system performance or utilization information via counters found in various /proc files; and 3) ensure Watt meter log files and all /proc file system log files are aligned (calibrated or synchronized). Consistent sampling is defined as sampling that takes place consistently at the middle of each sampling period. In the current work, the sampling period is an integer measured in seconds, *T0_sec*. For both Watt meter and /proc sampling, the finest granularity common to both is one second, thus *T0_sec* = 1.

This chapter describes the methodology applied to sampling power load and /proc files on *virgo2* compute nodes. The next chapter discusses how samples are processed and aligned as well as how power models of compute nodes are formulated with Watt meter and system utilization information.

4.1 Virgo2 Cluster Specifications

Data collection and analysis was conducted on *virgo2*, a Beowulf class HPC cluster with a single frontend node and 19 compute nodes and one memory node (at the time of this writing, the cluster consisted of 20 compute nodes and 2 memory nodes but two were out of operation).

Table 16 shows *virgo2* node specifications. (Note: all tables and figures shown in this chapter are reprinted from **Appendix FT4**). All compute nodes were built with Tyan Tempest i5400XT motherboards (Tyan, 2003) and were configured identically with 2 Intel Xeon 4-core processors (a total of 8 cores per node) each with 8 GB RAM installed. One compute node has been modified to form a GPU node. The only difference between a "standard" compute node and the GPU node is the installation of an NVIDIA GeForce GTX 480 video card.[6] The memory node used a Tyan Tempest i5400PW S5397 motherboard with 64 GB memory. Finally, two compute nodes were configured with external *watts up? Pro* power meters.

Table 16 – Virgo2 cluster specifications.

Type	Nodes	CPUs	Cores	cpu MHz	Memory	Cache Size
Frontend	1	2	8	2000	8 GB	6144 KB
Compute (2 metered)	18	2	8	2000	8 GB	6144 KB
GPU	1	2	8	2000	8 GB	6144 KB
Memory	1	2	8	2000	64 GB	6144 KB

Each node included two Gigabit Ethernet interfaces (one onboard and one PCI), however only one interface was used for HPC computations, *eth0*. The cluster was interconnected with a single Cisco Catalyst 3750 switch. (Two additional switches were included for channel bonding experiments but were not utilized during the data collection phase of this research).

Virgo2 is an example of a power limited cluster. For this cluster, power is supplied by six 20 amp, 120 volt circuits with a theoretical maximum of 14,400 Watts for the entire cluster. In this case, power is clearly a primary design constraint that limits the size of the cluster. Each node was equipped with an Antec TP2-550EPS12VXR power supply rated at 550 Watts. This makes the cluster's theoretical maximum peak power draw (based on power supply rating) 11,550 Watts.

[6] Note that the video card GPUs did not participate in HPCC benchmark computations. Also note that the video card increases idle power consumption on the compute node.

The cluster was constructed using Rocks Cluster Distribution version 5. Rocks is an open-source Linux cluster distribution based on CentOS that allows end users to build and manage HPC clusters (Rocks, 2011). At the time of this writing, the cluster's peak HPL rating was around 700 GFlops (Gumataotao, 2010).

4.2 Measuring power consumption on compute nodes

Power consumption on compute nodes was measured using two *watts up? Pro* power meters (WU, 2008). One meter was connected to compute node c0-11 and another to c0-12 in a *daisy chain* or *inline* configuration (see **Figure 29**). In this configuration, power flows from the electrical outlet through the meter, where it is sampled, to the compute node power supply. From there, power is converted from AC to DC and is subsequently drawn by system components.

External power meters such as the *watts up? Pro* provide several advantages for measuring system level power consumption including:

- Set up is minimally intrusive. Using an external Watt meter does not require taking apart a compute node and using specialized equipment to measure power consumption.
- The meters are affordable (the cost at the time of this writing for each meter was $195).
- Meter operation is facilitated by GUI software which enables quick access to the meter settings and a simple interface for downloading results.
- Data is downloaded from the meter to a PC using a USB connection and is saved as flat ASCII text files. This means data can be easily imported to a spreadsheet application or read by software packages such as R and Matlab.
- The meter supports tracking up to 16 variables and a timestamp. Variables tracked include: Watts, Volts, Amps, Watt Hours, Average KiloWatt hours, Cost, Power

Factor, Duty Cycle, Power Cycle, as well as minimum and maximum values for Watts, Volts, and Amps.

- Samples are taken *consistently* each sample period. This means that samples are rarely (if ever) skipped and thus a complete sample set is obtained for each trial.

The *watts up? Pro* operates by sampling power consumption as it passes through the meter to the compute node. The meter's primary sampling features include:

- The sampling interval ranges from 1 second to 24 hours.
- Samples are stored in internal memory.
- The total number of samples stored in memory depends on the number of variables or metrics tracked by the meter. If only one metric is selected, such as Watts, internal memory can hold up to 32,000 samples.
- The meter does not have a real time clock. Instead of time stamping each sample, sampling intervals are reported and the GUI software adds timestamps relative to the system clock the software is running on and the last record of the data set.[7]
- Because data from the Watt meter is downloaded to a PC running Windows XP, timestamps are based on the time since January 1, 1900. The timestamps have the format *days.value*, where value corresponds to the hours, minutes and seconds since January 1, 1900.
- Analysis of timestamps reveals the Watts up? Pro meters are sampling near the midpoint between sampling intervals.
- Note that /proc samples use Unix/Linux timestamps. Thus, to compare *watts up? Pro* timestamps to Linux based timestamps it is necessary to convert *watts up? Pro* to Unix time or vice versa.

[7] https://www.wattsupmeters.com/forum/index.php?topic=133.0

Watt meter limitations. The *watts up? Pro* places limits on data sampling granularity. In this research, a 1 second sampling interval was chosen to provide as much detail as possible to describe power consumption related to system utilization.

As it relates to data storage, HPCC benchmark runs were limited to approximately 8.9 hours maximum based on the internal memory limit of the Watt meters (32,000 samples at 1 sample per second translates to approximately 8.88 hours). The storage limit placed a limit on duration of trials primarily because the meter stops storing samples in memory while data is being downloaded. Because the *watts up? Pro* only supports downloading data from a relatively slow USB connection, downloads can take up to 4 minutes when memory is full. This means the meter loses approximately 240 samples each time data is downloaded and has the effect of compressing Watt meter data relative to /proc file system samples. Thus, if data is downloaded during a benchmark run, or between benchmark runs, Watt meter and /proc file samples do not align properly (data alignment and its impact on power models is described in greater detail in **Section 4.3.2**).

4.3 Measuring system utilization on compute nodes with HPCC

System utilization was measured by 1) launching an instance of the HPCC benchmark suite and 2) sampling the /proc files identified in **Table 2** in **Appendix FT3**. (OpenMPI, 2012) was used to execute the HPCC benchmarks for a specified number of nodes/cores and a C program was written to perform sampling and logging of counters and measurements. The HPCC benchmarks are discussed briefly in the next section. The primary features of the C /proc sampling program (proc_sample.c) include:

- Command line arguments that specify, among other options, the sampling interval (in seconds or microseconds), the /proc file to sample, and the log file that corresponds with the /proc file. In the research presented here, a one second sampling interval was selected for all computations run on the cluster. This

corresponds with the minimum sampling interval of the *watts up? Pro*. (Note: the C program allows specifying sampling intervals ranging from 1 microsecond, *usec* to several days in seconds, *sec*. In practice, sampling at a 1 microsecond interval may not be possible because loop overhead may be greater than 1 microsecond.)

- The main loop of the program ensures that samples are taken consistently each sampling period. This process is explained in greater detail in **Section 4.3.1**.
- The program uses the Linux system call `fread` to read an entire /proc file then applies the `scanf` system call to formatting strings in order to parse counters and extract measurements. All extracted values are mapped to a $1 \times N$ vector (where N is the total number of counters tracked by the /proc file). This vector is stored in a `malloc` buffer. This /proc file linearization is explained in greater detail in **Section 4.3.2**. More information on `fread`, `scanf`, and `malloc` can be obtained from the following manual pages respectively: `fread(3)`, `scanf(3)`, and `malloc(3)`.
- One instance of the program is spawned for each /proc file shown in **Table 2** and for each compute node participating in an HPCC benchmark run. This means a total of ten proc_sample.c programs are asynchronously sampling each node, with ten unique log files generated per node. For full cluster computations, this means a total of 200 log files are generated per benchmark run. See Section 4. for further discussion on the sampling process and its impact on node performance.

While the /proc file system is a pseudo file system, files in the directory can be read using the Linux system calls `fopen` and `fread`. **Figure 12** shows a template of the main /proc sampling loop. When `proc_sample.c` opens a /proc file, the file remains open for the duration of the sampling process. This avoids having to issue `fopen` and `close` each time a file is read or sampled thereby reducing overhead and increasing overall performance. One problem with this approach, however, is that the Linux system call `rewind` (reference the `fseek(3)` manual page for more information) does not reset the file descriptor pointer (`proc_fd`) to the beginning of the

open /proc file. This causes `fread` to sample the same data in the read buffer over and over, producing incorrect measurements of the counters. To fix this, a call to `setvbuf` with the `_IONBF` flag is made after the /proc file is opened (reference `setvbuf(3)` for more information). This tells the system not to buffer reads for the open file, and ensures `rewind` actually resets the position of the open file descriptor.

```
FILE * proc_fd = fopen(proc_file, "r");
setvbuf(proc_fd, NULL, _IONBF, 0);
char proc_buf [MAX_BUF_SIZE];
while(1){
  rewind(proc_fd);
  nread   =   fread(&proc_buf,   sizeof(char),   MAX_BUF_SIZE,
proc_fd);
  sleep(T0_sec);
 }
```

Figure 12 – proc_sample.c main sampling loop.

4.3.1 Description of the High Performance Computing Challenge (HPCC) Benchmarks

The High Performance Computing Challenge benchmarks (also referred to as the HPC Challenge benchmarks) are a suite of seven benchmarks designed to stress a different part of the memory hierarchy (Luszczek et al., 2005). HPCC provides a snapshot "of a system's performance bounds for real applications" (Song et al., 2009). The HPCC benchmarks were chosen for this research on this ground to identify power consumption bounds for entire clusters. The HPCC benchmark suite includes:

- RandomAccess (RA)
- Parallel Transpose (PTRANS)
- Double precision General Matrix Multiply (DGEMM)
- STREAM
- Fast Fourier Transform (FFTE)
- Latency/Bandwidth (L/BW or b_eff)
- High performance Linpack (HPL)

RandomAccess measures the *rate of random memory updates*. PTRANS measures the *rate of transfer* for larges arrays of data from multiprocessor's memory. STREAM is a benchmark that measures *sustainable memory bandwidth* (in GB/s) and Latency/Bandwidth measures *latency and bandwidth of communication patterns* of increasing complexity between as many nodes as possible. Bandwidth measurements are made during non-simultaneous (ping-pong) and simultaneous (random and natural ring pattern) communication and therefore *cover two complementary levels of contention*: no contention and contention caused when each process communicates with a randomly chosen neighbor in parallel (hpcc-faq, 2012). HPL, DGEMM and FFTE stress the *floating point performance* of a system using linear programs, double precision matrix multiplication and Fast Fourier Transforms respectively.

Four memory access patterns characterize the HPCC benchmarks based on low or high spatial and temporal locality (Song et al., 2009). **Table 17** shows each benchmark, its spatial and temporal locality, its mode (described next), and what it is designed to measure. Three modes distinguish HPCC environment configurations (Song et al., 2009):

- MPI (Global) – the benchmark runs in parallel across all processors (with communication)
- Star – the benchmark runs independently on all processors (no communication)
- Single (Local) – the benchmark runs on a single processor

Table 17 – Characteristics of the HPCC benchmarks.

Benchmark	Spatial	Temporal	Measures	Units
RA	low	low	Random updates to integer memory	GUP/s
PTRANS	high	low	Data transfer rate	GB/s
DGEMM	high	high	Floating point performance	FLOPS
STREAM	high	low	Memory Bandwidth	GB/s
FFTE	low	high	Floating point and memory transfer performance	FLOPS
L/BW	low	low	Latency and bandwidth of communication patterns	msec, MB/s
HPL	high	high	Floating point performance	FLOPS

In the current work, results for the HPCC benchmarks are collected in a file named `hpccout.txt` for the following benchmark/mode configurations:

1. MPI RandomAccess
2. StarRandomAccess
3. SingleRandomAccess
4. MPI RandomAccess LCG
5. StarRandomAccess LCG
6. SingleRandomAccess LCG
7. StarDGEMM
8. SingleDGEMM
9. StarSTREAM
10. SingleSTREAM
11. MPIFFT
12. StarFFT
13. SingleFFT
14. Latency/Bandwidth
15. HPL

Note: LCG indicates the use of linear congruential generator (LCG) for random number generation. Also note that PTRANS is not included in results presented in **Chapter 6**.

Figures 13 through **18** show power load (in Watts) for split executions of the HPC Challenge benchmarks (minus PTRANS) for 1, 4, 8, 16, 32 and 160 cores respectively (See **Chapter 6** for details on split HPCC computations). Note that HPCC power footprints for more than eight cores tend to mirror the pattern shown in **Figure 15** for eight cores. (See **Section 6.6** for a description of each phase of the HPCC benchmarks).

4.3.2 Detailed description of the C sampling program

4.3.2.1 Under sampling /proc files

Figure 19 illustrates the how the main sampling loop in `proc_sample.c` was originally coded. X_k denotes the time a read actually occurs (i.e. the time the *k*th sample is taken). As shown, the loop uses the system call `sleep(T0_sec)` as a "door-stop" before taking the next sample. The sampling process is illustrated in **Figure 20**. Here, T_k denotes the start of the *k*th sampling interval for which the *k*th data value is ready to be sampled (ideally at the midpoint between T_k and T_{k+1}). As with **Figure 19,** X_k denotes the time a read actually occurs (i.e. the time the *k*th sample is taken) where the blue dot shows exactly where on the time line the sample is taken.

Now, suppose *loop_OH* denotes the sum of all the time spent executing the code between the `read` and `sleep` system calls shown in **Figure 19**. After taking a sample (i.e. after data is read into the program's internal memory from /proc), when program execution reaches `sleep(T0_sec)`, the next sample will be taken at (*loop_OH* + *T0_sec*). This is illustrated in **Figure 20** where the blue arrow corresponds to *loop_OH* and the black arrow corresponds to the length of the sampling period (*T0_sec*). As shown, sampling intervals are "skipped". That is, there exist sampling intervals where a sample is not taken.

As depicted in **Figure 20**, data value X_{k+1} is sampled late and that data value X_{k+2} does not occur within its corresponding sample interval, T_{k+2}. As a result the data set is under sampled and does not align properly with respect to the power meter values. This causes distorted models of power consumption. Note that the figure also shows samples are not consistently taken at the middle of each sampling interval.

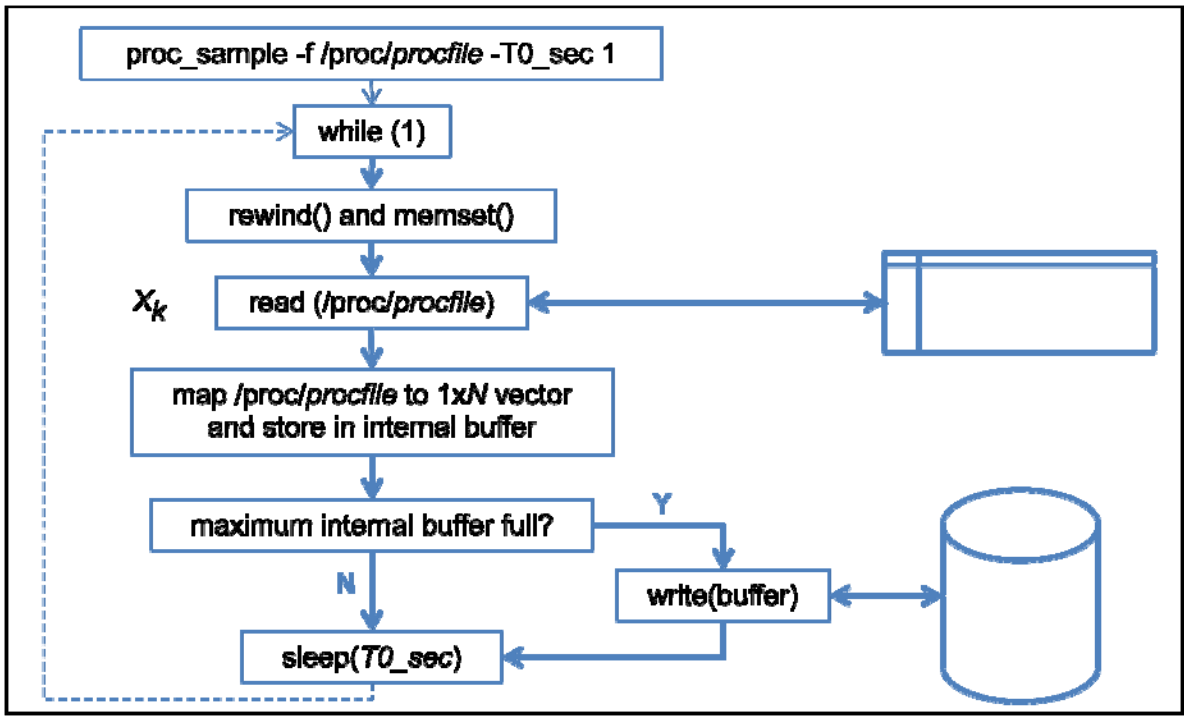


Figure 19 - Initial version of main loop for proc_sample.c.

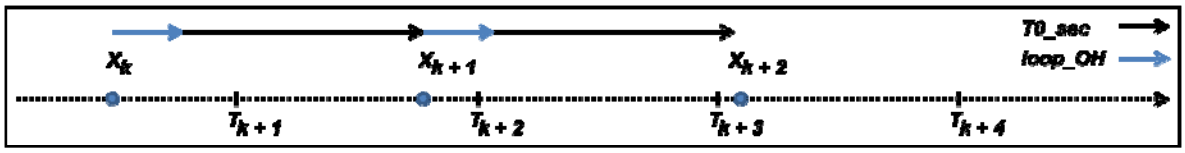


Figure 20 - Skipped sample in sampling interval $\{t_{k+2}, t_{k+3}\}$.

4.3.2.2 Consistently sampling /proc files

Figure 21 shows how /proc files should ideally be sampled. Starting with sample ***k*** at time X_k, the next sample should be taken at time X_{k+1}. To do so, the program should sleep for an amount of time equivalent to (*T0_usec – loop_OH*) represented by the green arrow in the figure.

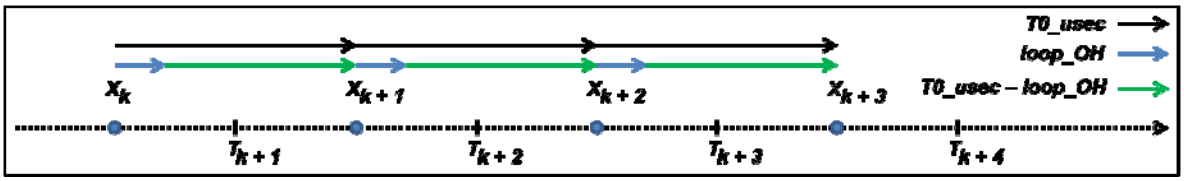


Figure 21 – Ideal sampling.

One attempt to correct the under sampling problem was to take the sampling interval (*T0_sec*), convert it to micro-seconds (*T0_usec*), and replace the `sleep(T0_sec)` system call in **Figure 19** with `usleep(T0_usec – loop_OH)`.[8] There are a few problems with this approach however.

First, assume that $\boldsymbol{X_k}$ occurs at the middle of the current sampling period $\boldsymbol{T_k}$. The next sample should be taken at time $\boldsymbol{X_k}$ + (*T0_usec – loop_OH*). Ideally this should result in samples taken consistently at the middle of each subsequent sampling period. In practice, however, sample times tend to creep or drift away from the midpoints and eventually some sample periods are skipped while other sample periods may include more than one sample. The end result is that sometimes /proc samples will be oversampled while at other times they can be under sampled. In other words, /proc files are not consistently sampled.

The problem described above occurs because the algorithm in **Figure 19** does not take advantage of information that specifies the exact (system) time at which samples are being taken relative to the midpoint of the sample interval. The final version of the main loop capitalizes on this observation by considering the time of occurrence of the next `sleep` system call relative to the time the current sample is taken or /proc file is read. After the current sample is taken, the main loop estimates when the next sample should be taken based on the current sample timestamp. The final sampling loop is shown in **Figure 22**. When program execution reaches `usleep(T0_usleep)`, the process sleeps for the amount of time necessary to ensure the next sample occurs at any time before or near the midpoint of the next sample interval.

[8] Note that the Linux system call `sleep` is constrained to sleep for integer periods of time. On the other hand, `usleep` enables sleeping for time granularities that include microseconds.

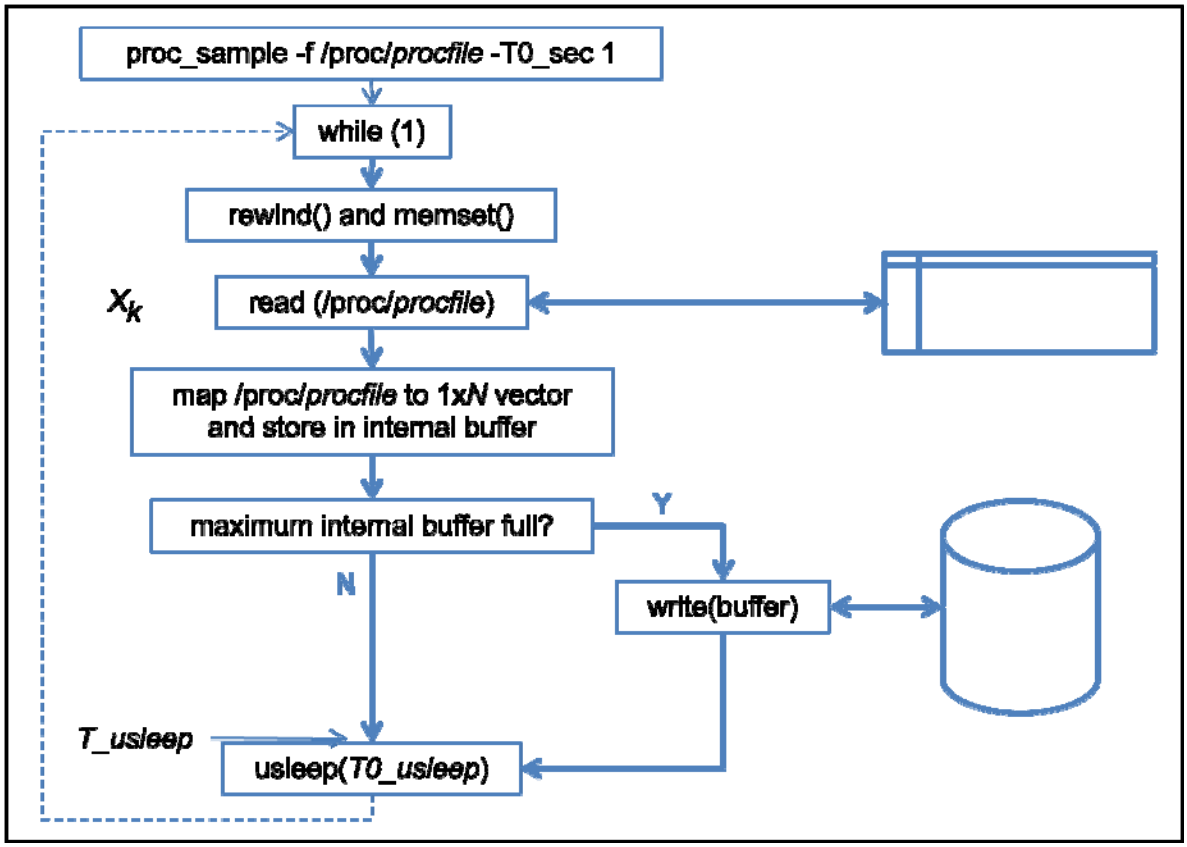


Figure 22 – Final sampling loop.

Note there are a few details not captured by **Figure 22**. First, how does the algorithm initially know when to take the first sample so that it occurs at the midpoint of the interval? This is done by inserting the code shown in **Figure 23** just before entering the while loop. The code basically states, if the current time is before the midpoint of the current sampling interval (500,000 usec minus a fudge factor of 100), sleep until the midpoint arrives. Otherwise, immediately take a sample because the midpoint was passed. In the latter situation, the sample is taken late in the sampling interval.

```
gettimeofday( &tv1, NULL);
if ( tv1.tv_usec < 499900 )
  usleep( 499900 - tv1.tv_usec );
```

Figure 23 – Wait for the midpoint of the sample period or immediately proceed.

After taking the current sample ***k***, processing the sample, and checking whether internal memory is full, a timestamp (*T_usleep*) is taken immediately before the process sleeps at `usleep` (**Figure 22**). The following code (**Figure 24**) is then used to estimate when the next sample should occur and for how much time (*T0_usleep*) the process should sleep to ensure the next sample, ***k*+1**, occurs at the proper time.

```
gettimeofday( & Time_xk, NULL);
if (tv2.tv_usec > 500000)
  T0_usleep = T0_usec - (tv2.tv_usec - 500000);
else
  T0_usleep = T0_usec;
```

Figure 24 – Estimate the time when the next sample should be taken.

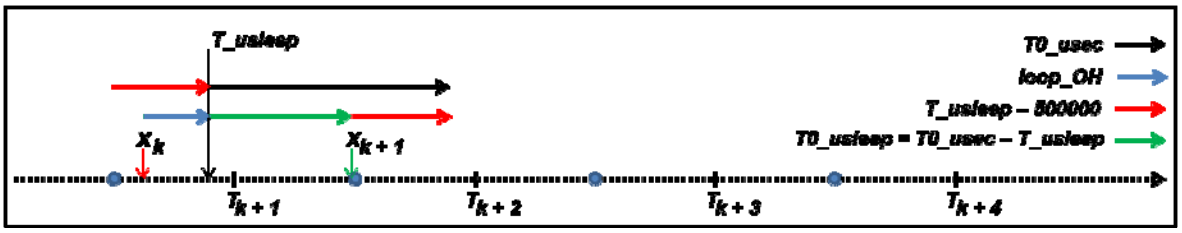


Figure 25 – Determining the sleep duration of the sampling process.

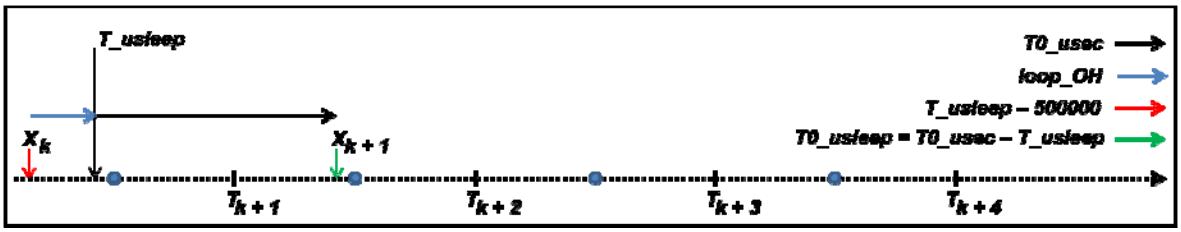


Figure 26 – Determining the sleep duration of the sampling process.

The sampling loop then works as shown in **Figure 25**. If timestamp *T_usleep* is taken after the midpoint of the current sample period (black vertical arrow), take the next sample near the midpoint (indicated with a blue dot on the number line) of the next sample period (green vertical arrow). Otherwise, if *T_usleep* occurs before the midpoint of the current sample period (**Figure 26**), sleep for a period equal to sample period (*T0_usec*) before taking the next sample. This process ensures samples are taken any time before the midpoint of the next sample period. Any time the current sample is taken after the midpoint of the current sample period, the sampling times are re-calibrated to occur near the midpoint.

The advantages of this approach are ease of implementation, reduced overhead (i.e. there is no need to calculate *loop_OH* and subsequently estimate the sleep duration), and if samples are skipped, the loop will automatically recalibrate. The primary disadvantage is that some samples may skip and the algorithm doesn't do anything to compensate for missing samples.

One thing to note is that it is nearly impossible to predict the duration of *loop_OH*. In theory, *loop_OH* is very small given disk writes are kept reasonably small (the next sub-section discusses disk writes in more detail). However *loop_OH* duration can result in skipped samples. This is because many factors contribute to the duration of overhead, including number of processes running on the system, time spent reading /proc files, time spent waiting for IO (e.g. writing internally buffered samples to log files), whether or not other processes will hog all cpu cores, etc. Any time *loop_OH* > *T0_usec*, samples will be skipped. The following example shows another case for which samples are skipped.

Figure 27 shows a worst case scenario where X_k is taken sufficiently close to the start of the next sample period that *loop_OH* causes the *T_usleep* timestamp to occur in the next sample period. In this case the process sleeps for *T0_usec*, overshooting the next sample period, T_{k+1}. In practice however, the loop illustrated in **Figure 22** keeps samples close to the midpoint and few samples are skipped. Future work can address this issue by testing to ensure both X_k and *T_usleep* occur within the same sample period so that the next sample is scheduled to be taken at the proper time.

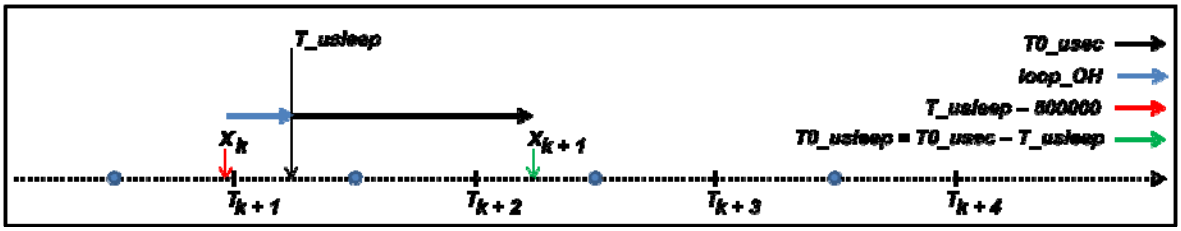


Figure 27 – Worst case scenario where a sample is skipped.

4.3.2.3 Other sampling considerations

The condition *loop_OH* > *T0_usec* can occur when writing to the log file. This will typically occur if the internal buffer is sufficiently large that disk writes exceed *T0_usec*. This condition is avoided by keeping the internal buffer reasonably sized to ensure that a sufficient number of samples are stored to make writing log files efficient yet not too large to cause samples to be skipped.

Finally, in the worst case, because the sampling program voluntarily sleeps between samples, there is a possibility that system delays may cause *loop_OH* to be greater than *T0_usec*. In such cases, samples will be lost due to system activity. At the time of this writing nothing is done to account for these missed samples because in this work, very few samples are skipped.

4.3.3 Mapping /proc files to 1×N vectors

As indicated in **Chapter 3**, the formats of proc files vary widely. Let $\boldsymbol{N_p}$ denote the number of variables to track in proc file $\boldsymbol{p}$. Then basic proc file formats include:

- A $1 \times \boldsymbol{N_p}$ set of values separated by white space.
- An $\boldsymbol{N_p} \times 1$ set of values separated by line breaks.
- An $m \times n$ set of values, where $\boldsymbol{N_p} = m \times n$, m is the number of rows, and n is the number of columns.
- Hybrid combination of basic formats.

To further complicate things, each proc file may have version numbers, one or more column headers, row labels, special characters (e.g. %), etc.

To simplify data collection, each proc file is "linearized" so that samples can be collected and analyzed in a consistent manner. Linearization is accomplished by defining a `scanf` format string for each /proc file that filters out unnecessary information (such as version information, column headers, and special characters). Specifically, the sampling and processing of a proc file is accomplished by:

1. Reading the entire proc file into a character buffer (array) using `fread.`
2. Parsing the character buffer using the predefined format string to extract counters/values.
3. Storing values into an appropriately typed (i.e. long integer, double, etc.) and malloc'ed buffer as a single record.
4. Time stamping each record.

Note: for proc files that contain a mix of integer and floating point values (such as /proc/loadavg), integer values are converted to double values and subsequently stored in

malloc'ed memory. This implies that there are two types of log files, those that contain `long integer` types and those that contain `double` types.

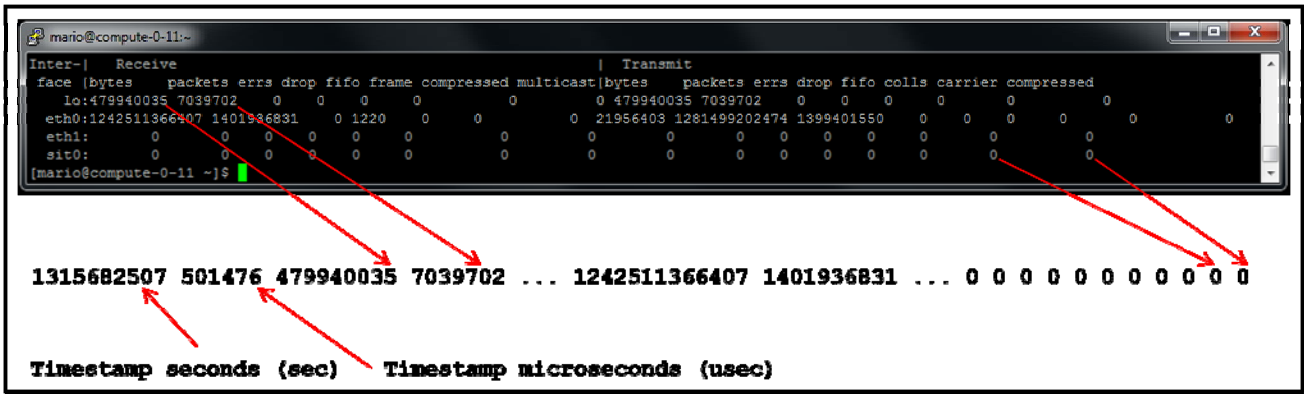


Figure 28 – Mapping /proc/net/dev to a 1 × *N* vector with timestamps.

Table 18 – Sample log file showing /proc/net/dev samples.

		lo		eth0		eth1		sit0	
sec	usec	Rx Bytes		Rx Bytes		Rx Bytes		Rx Bytes	
1315682507	501092	73894801	…	1234808192764	…	0	…	0	…
1315682508	501021	73894801	…	1234808192764	…	0	…	0	…
1315682509	502077	73894957	…	1234808247046	…	0	…	0	…
1315682510	501051	73894957	…	1234808247046	…	0	…	0	…
1315682511	501117	73895123	…	1234808248724	…	0	…	0	…
1315682512	501172	73895123	…	1234808248724	…	0	…	0	…
1315682513	501231	73895123	…	1234808250308	…	0	…	0	…
1315682514	501380	73895123	…	1234808250308	…	0	…	0	…
1315682515	501352	73895123	…	1234808250308	…	0	…	0	…
1315682516	501418	73895289	…	1234808250308	…	0	…	0	…
1315682517	500471	73895289	…	1234808250308	…	0	…	0	…
1315682518	500533	73895289	…	1234808250308	…	0	…	0	…
1315682519	500592	73895289	…	1234808250308	…	0	…	0	…
1315682520	500653	73895455	…	1234808250308	…	0	…	0	…

Figure 28 illustrates the linearization process applied to one sample of /proc/net/dev. **Table 18** shows what a /proc/net/dev resulting log file looks like (note how sample times are consistently taken near the midpoint of the sample period indicated by a usec value near 500,000). In **Table 18**, each column corresponds to utilization metrics of a specific /proc file counter. In this dissertation, each individual /proc counter column is treated as a variable. For this example the variables are identified by the column labels.

4.4 Running and sampling system activity on the full cluster

As noted earlier, for each compute node involved in a benchmark run, the ten proc files identified in **Table 2** were sampled each sample period (*T0_sec*) by a separate proc_sample.c process. This means for full cluster analysis, 200 sampling processes are asynchronously sampling 200 proc files, with an additional two power meters asynchronously sampling power consumption on two of those nodes.

While this approach has its drawbacks, it does simplify the program's code and its operation. It can be shown that for sampling intervals greater or equal to 1 second, the impact on power consumption and performance is minimal. In addition, careful selection of internal buffer sizes ensures that each instance of the program maintains a small memory foot print. Note, however that there is a tradeoff between data logging and memory consumption (buffer size). The smaller the buffer, the more frequently writes to disk occur, with very small internal buffers resulting in inefficient writes. On the other hand, memory sizes that are too large have the drawback of consuming valuable memory and wasting a significant amount of time writing to log files. Long writes have two implications, power spikes associated with writing to log files and potential negative impact on consistent sampling (i.e. samples can be skipped during long writes). For the purposes of this research, each C sampling program instance stored 100 samples before writing data to disk.

Also as noted above, *T0_sec* was chosen to be one second, which corresponds with the finest sampling granularity of a *watts up? Pro* meter. This means, as indicated in **Table 2**, a total of 2,927 variables or utilization counters were sampled each second for each compute node. For full cluster analysis, this results in a total of 2,927 * 20 nodes = 58,540 utilization counters sampled each second. Because a *watts up? Pro* meter can track as many as 32,000 samples before losing information, this means full cluster analysis can generate log files that in total contain nearly 1.87 billion counter values (58,540 * 32,000 = 1,873,280,000) when the log file corresponds exactly with a full set of Watt meter data, approximately 8.53 hours worth of Watt samples, assuming Watts is the only metric tracked. Note that log files can contain vastly more data, depending on how long the sampling program, `proc_sample.c`, has been collecting data samples from the system.

The next chapter (5) discusses how this data is filtered and aligned before analysis followed by a discussion of how power models are formulated and the R power modeling tool.

4.5 Chapter Summary

Power meter and system utilization samples are collected from *virgo2.ece.utep.edu*, a 21 node power limited HPC cluster. This chapter shows collecting utilization samples from the /proc file system can be accomplished with standard Linux system calls such as `fopen`, `fread`, and `rewind`. However, this requires extracting values from non-uniformly formatted pseudo-files. To facilitate generation of log files, data contained in the various /proc files are "linearized". That is, each /proc file is read into the sampling program's internal memory. Targeted values are parsed from the input and then mapped to a single record that is written to the next row of a log file.

A method for data sampling is described that ensures samples are collected in a consistent manner at the middle of each sampling interval. As indicated in **Chapter 3**, a total of 10 files are selected for final analysis including buddyinfo, diskstats, interrupts, loadavg, meminfo, net/dev,

schedstat, slablinfo, stat, and vmstat. Within these files 2,927 variables are extracted from the files per sample interval per compute node. This means Watt meter log files can contain up to 8.53 hours worth of Watt meter data with /proc files containing up to two billion utilization counter values when 32,000 samples are taken across 20 compute nodes, with each node logging data from 10 /proc files. This implies a need to eliminate redundant information and information that is not useful for modeling and analysis. The methods implemented to reduce the amount of data are covered in the next chapter.

Chapter 5: Power modeling methodology

As indicated in **Chapter 4**, the procedure for sampling /proc files during an HPCC computation is implemented using a C program. Power modeling and analysis, on the other hand, is conducted "offline" with R, a programming language and software environment for statistical computing (R, 2012). R is suitable for the current research for the following reasons:

- It provides a power graphical environment with simple interactive controls provided by the rpanel package.
- R libraries can be easily obtained and installed, including the linear programming libraries that are the backbone of the least squares models developed in this paper.
- It is published under the GNU General Public License and is freely available for download.

The following steps illustrate how the modeling environment works:

1. Cumulative counters from /proc log files are converted into activity per sample counters.
2. Counters in /proc log files that are constant are identified and eliminated from consideration.
3. Idle power is determined for each metered node. Idle power is the power load of a system that is not busy but is ready and waiting to do work. In the current work, idle power is determined by taking the minimum power load of an idling compute node.
4. Next, "active" power vectors are calculated for each node by subtracting idle power from sampled power. Active power (not to be confused with active power in an AC circuit) is power consumed by a system that results from an increase in system activity. Active power is the power load above a system's idle power load. Note: subtracting power samples by the minimum value of an idling system versus the average value safeguards against negative Watts appearing in active power vectors.

5. Processed proc files (steps 1 and 2 above) are aligned with active power vectors. An automatic alignment procedure is implemented based on a /proc variable that is representative of system activity during benchmark calculations.
6. Using interactive controls, least squares minimization equations are manually or automatically formulated and coefficients are calculated for resulting models.

The remainder of this chapter provides details as it relates to each of the steps above. In particular, the following topics are covered: processing raw proc samples, Watt meter and /proc sample alignment algorithm, formulating least squares minimization equations, implementation of the least squares minimization equations in R, and a brief discussion of the R interactive modeling tool developed in the current work. Finally, two key analysis tools are described because they are implemented in R: Coefficient of Determination and Factor Analysis.

5.1 Why use the R programming language?

R is a programming language and software environment for statistical computing (R, 2012). Although R is geared toward statistical computing, it is well suited for the research conducted here. One of the primary reasons to adopt R is because it is available under the GNU General Public License and is freely downloadable. This makes R an inexpensive alternative to other packages such as MATLAB. Other key features of R that apply to this research include:

- General matrix arithmetic tools.
- Linear programming libraries.
- Powerful graphics and visualization tools.
- Simple interactive controls.
- A well developed and simple programming language that supports scripting.
- Easily extended via packages: at the time of this writing over 3,670 packages were available through the Comprehensive R Archive Network (CRAN, 2012).

In this research R scripts were written to perform all data processing, power modeling, and analysis. An interactive framework was developed to simplify development of HPC cluster power models.

Cluster power modeling and analysis. Cluster power modeling and analysis in R was conducted after data samples were collected for a particular benchmark run. **Figure 29** illustrates the data collection and analysis framework. Note that all figures and tables in this chapter are reprinted from **Appendix FT5**. The figure shows:

- The C sampling programs are launched on compute nodes and mark the official start of a trial. The C sampling programs are launched before HPCC benchmarks are calculated and continue sampling for a period of time after the HPCC benchmarks complete. Sampling occurs on each compute node that participates in HPCC computations for this trial.
- Watt meters continuously sample once they are plugged in to a power outlet. To mark a trial, each meter's internal memory is cleared sometime after the C sampling programs are launched. The meter is allowed to collect samples for several minutes before HPCC benchmarks are executed. The meters continue sampling for a period of time after the HPCC benchmarks complete then data is offloaded to the external PC hard drive via a USB connection. Meter data is typically downloaded before C sampling programs terminate.
- The compute nodes, unlike the Watt meters, periodically write data to separate log files residing in an NSF directory. Note that the compute nodes are capable of collecting data indefinitely (limited only by the amount of disk space available on the NFS node).
- Once data collection is complete, a separate computer system is used to read data into the R environment from the two primary data sources, the external PC and the NSF node.

- Once data is available in the R environment, R scripts process and align data samples. The following sections provide details on these two steps.

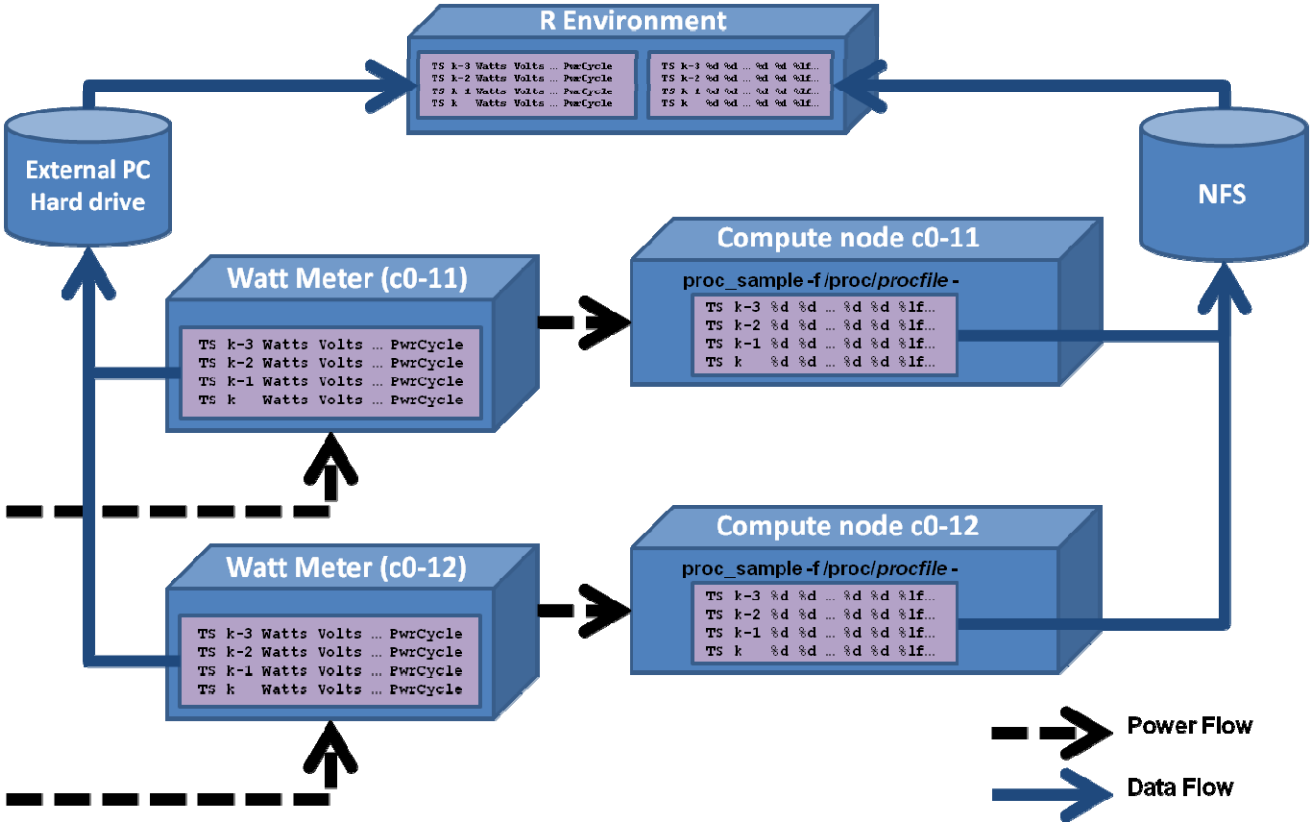


Figure 29 – Power Modeling and Analysis Environment.

5.2 Processing log files

Processing occurs before Watt meter and /proc samples are aligned. This section discusses the nature of /proc file variables, required processing steps, how the data is filtered to eliminate irrelevant information, and processing applied to Watt meter data.

5.2.1 Processing /proc files

As indicated in **Chapter 3**, /proc values can be fixed metrics such as the amount of memory installed on a node. Alternatively, /proc values can be *cumulative counts* (aggregates) of system activity since last boot or a measurement of the *level of usage* between successive reads to a /proc file. Examples of cumulative counters include the amount of time spent by the system executing in user or system mode since last reboot and the number of packets transmitted or received since last reboot. Examples of *level of usage* measurements include number of currently allocated pages, number of dirty pages, and number of active kmem_cache objects. In the current work, *level of usage* counters can be used in models without further processing. However, to be meaningful, cumulative counters need to be converted to values that describe the amount of activity that occurs per sample. For instance, the number of packets transmitted since last reboot need to be converted to the number of packets transmitted per sample (number of packets between successive /proc reads). Then, constant values need to be eliminated from the data set because they do not provide meaningful system utilization information. R scripts carry out these two steps as follows:

- First, cumulative counters are converted to reflect activity per sample by taking the difference between successive samples and storing the data as a separate vector in the R environment. In the current work, identifying cumulative counters involved individually analyzing each variable or reading /proc related documentation. Mathematically, cumulative counters are converted using the following procedure. Let $\boldsymbol{V}$ be a column vector that contains a set of samples for a cumulative counter and v_k and v_{k+1} denote two successive values or elements of $\boldsymbol{V}$, where k denotes the kth sample value. A new column vector $\boldsymbol{V'}$ of activity per sample values is generated such that $v'_k = v_{k+1} - v_k$, for $k = 1 \ldots (N - 1)$, with $v'_k = 0$ for $k = N$. The Nth element of V' is set to 0 because initially, $\boldsymbol{V'}$ has $(N - 1)$ elements. To ensure $\boldsymbol{V'}$ has N elements a zero value is appended to the vector.

- Second, note that a variable's samples are constant if the standard deviation of the samples is 0. After cumulative counters are converted, all variables whose standard deviation is 0 are identified and removed from analysis. All variables that are not eliminated by this step are referred to as "*surviving*" variables and correspond to data samples that reflect some type of system activity.

Note that in R, $\boldsymbol{V'}$ can be calculated as follows: Let $\boldsymbol{V_P}$ denote a cumulative counter variable that measures system activity in /proc file $\boldsymbol{P}$. Activity per sample, $\boldsymbol{V'_P}$, is determined by taking the difference between two ranges of $\boldsymbol{V_P}$ as follows: $\boldsymbol{V_P}[2{:}N] - \boldsymbol{V_P}[1{:}N{-}1]$. $\boldsymbol{V'_P}$ is now an activity per sample vector with $N - 1$ elements, so a zero value is appended $\boldsymbol{V'_P}$. This ensures that the number of elements (cardinality) does not conflict with other log file variables that do not need to be converted to activity per sample.

5.2.2 Surviving variables and target variables

Of the total number of variables logged during the data collection phase, the elimination step eliminates approximately 2,000. This means after processing, the number of *surviving* variables is about 850. **Table 19** (reprinted below) shows the number of *surviving* variables per /proc file that correspond to an HPCC computation executed on a single processor and core on one compute node (1-node/1-processor/1-core).

A more in-depth analysis of *surviving* variables reveals that many, while their standard deviation is non-zero, are nearly constant for an entire sample set of values. This implies that they are not closely linked to system activity and subsequently power consumption. This leads to the selection of "*target"* variables. A final list of *target* variables was selected based on 1) whether or not the variable can be linked to power consumption; 2) whether the variable could be meaningfully applied to general cluster power consumption models and 3) whether including the

variable in models provides meaningful information as it relates to system utilization. The final *target* variables per /proc file are shown in **Table 14** and **Table 15** in **Appendix FT3**.

Table 19 – Number of surviving variables per proc file.

/proc file	Counters	Surviving	Target
buddyinfo	33	20	16
diskstats	385	23	11
interrupts	138	20	20
loadavg	5	5	5
meminfo	30	16	14
net/dev	80	9	8
schedstat	672	416	24
slabinfo	1,216	259	31
stat	317	56	15
vmstat	51	35	18
Total	**2,927**	**859**	**162**

It is important to point out there is not necessarily a one to one relationship among *surviving* and *target* variables across compute nodes. That is, some /proc variables show system activity on one node that may be constant or nearly constant on another. This implies that power models may include variables that are unique to different nodes. Further, power models may produce variables that are unique to different benchmark computations (e.g. a 16 core computation compared to a 160 core computation). This complicates the development of general power models and is discussed in greater detail in **Chapter 6**.

Finally, note that in the current work, *target* variables are a subset of *surviving* variables, but need not be so. The analysis framework developed in this research allows *target* variables to be easily included or removed from the analysis. Thus at any time, should a significant variable be identified, it can be added to the R analysis framework and incorporated into power models. This is currently done by editing an R script that identifies *target* variables. In addition, the R

framework allows for analysis of *surviving* variables as well as the entire set of sampled variables. For the purposes of this research however, only *targeted* variables were considered.

5.2.3 Watt meter processing

Watt meter samples are not altered because they already represent power load per sample. However a second column, referred to as "*Watts_delta*", is added to the R environment. *Watts_delta* corresponds to *active power* (as discussed in **Section 5.1**) and is calculated by subtracting Watt meter samples by the metered compute node's minimum power load when the node is idling. This is done to ensure power models are based on similar types of measurements. Note that /proc activity per sample is characterized by two criteria: 1) values are greater or equal to 0; and 2) values near zero correspond to utilization of an idling compute node. *Watts_delta* values are also characterized by the two criteria as it relates to power. Thus, when a system enters a busy state, both system activity counters and power consumption increase and power models are thus based on similar types of measurements.

5.3 Watt meter and /proc sample data alignment

Watt meter data and *surviving* variables need to be aligned before constructing least squares models. This is because misaligned data impacts the accuracy of power models depending on the degree of misalignment. This section provides details on how data is aligned. In the following, the variable tracked by the /proc file is referenced by appending a dot "." followed by the variable name in italics to the absolute path to the /proc file. For instance /proc/stat.*user* references the variable "*user*" found in /proc/stat. Once a variable has been introduced, it is referred to by its italicized name to simplify reference to the variable.

5.3.1 Selecting an alignment variable

The first step in aligning data is to choose a /proc counter that strongly correlates with HPCC benchmark runs and power consumption. Based on trial-and-error, it can be shown that /proc/stat.*user*, /proc/stat.*sys*, /proc/meminfo.*commitA*, and /proc/vmstat.*pgfree* (among others) provide good alignment results. Note that each performs better in special cases over others. At the time of this writing, however, /proc/stat.*user* has provided the best overall results and is thus the default alignment variable.

5.3.2 Alignment algorithm

An R script is used to automatically align /proc utilization data with Watt meter data. Without the automatic script, aligning data would be tedious, time consuming, and error prone. The alignment algorithm is illustrated in **Figures 30** and **31** (reprinted below) and operates as follows:

1. Select a /proc utilization counter sample, $\boldsymbol{U_x}$, that will be aligned with Watt meter samples, $\boldsymbol{W}$. (x denotes the selected proc variable. In **Figures 30** and **31**, x = /proc/stat.*user*.)
2. Define three windows, the Data Window (*DW*), a Watt window (*WW*) and a sliding Window (*SW*). Note each window has an associated lower bound and an upper bound pairs $\{DW_{\text{lbn}}, DW_{\text{ubn}}\}$, $\{WW_{\text{lbn}}, WW_{\text{ubn}}\}$, $\{SW_{\text{lbn}}$, and $SW_{\text{ubn}}\}$.
3. Set *WW* bounds equal to the starting and ending positions of Watt meter samples (**Figure 30**). Note that *WW* bounds can include the entire Watt sample or a subset of the sample.
4. Calculate the Watt window size, $WW_{\text{size}} = WW_{\text{ubn}} - WW_{\text{lbn}} + 1$.
5. Set up *SW* bounds as follows: $SW_{\text{lbn}} = 1$ and $SW_{\text{ubn}} = WW_{\text{size}} - 1$.
6. Calculate $SW_{\text{size}} = SW_{\text{ubn}} - SW_{\text{lbn}} + 1$.

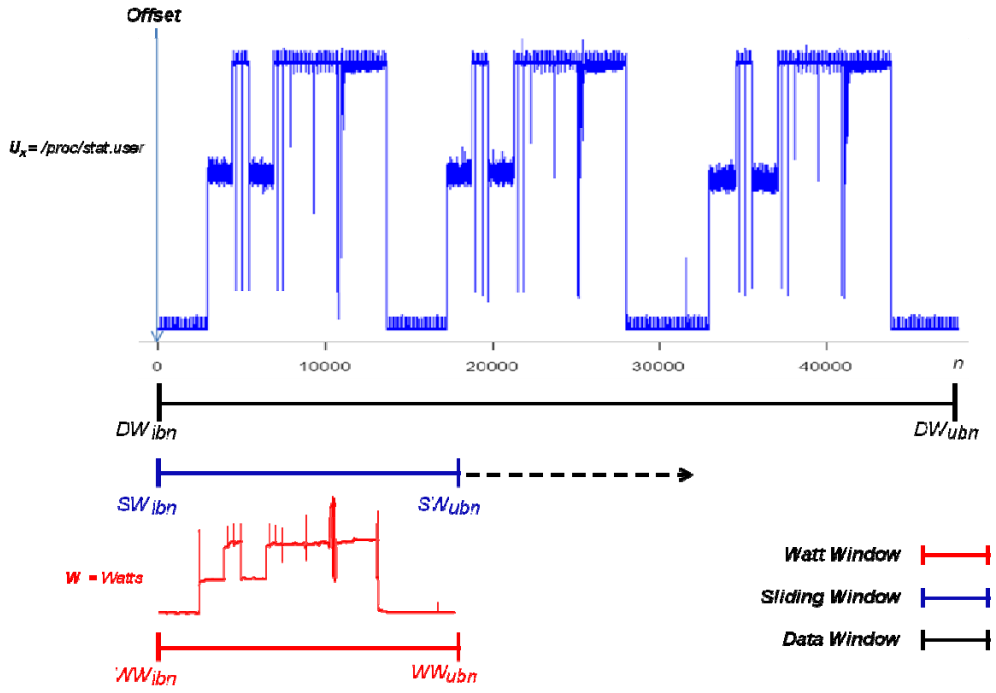


Figure 30 – Illustration of the alignment algorithm using /proc/stat.user samples..

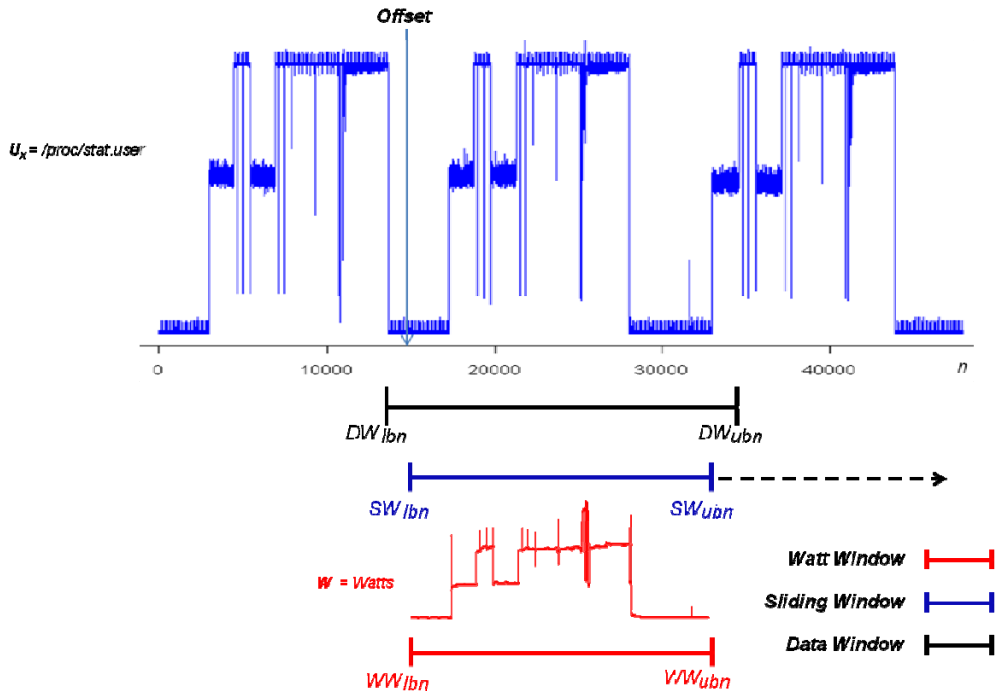


Figure 31 – Offset shows where Watt meter samples align with /proc/stat/user samples..

7. Set a value for a variable i such that $1 \leq 2^i << n$, where n is the number of /proc samples contained in $\boldsymbol{U_x}$. (The default value of i is 3).
8. Set $step\_size = 2^i$
9. While $SW_{ubn} \leq n$
    10. Correlate $\boldsymbol{SW}$ $[WW_{lbn}, WW_{ubn}]$ and $\boldsymbol{U_x}$ $[SW_{lbn}, SW_{ubn}]$
    11. If this correlation is greater than the maximum correlation so far, set $Offset = SW_{lbn}$
    12. Increment SW_{lbn} and SW_{ubn} by $step\_size$
13. Redefine DW to $\{Offset - KN*step\_size, Offset + SW_{size} + KN*step\_size\}$.
14. If $i > 0$, decrement i and return to step 7
15. Stop.

Notice that a step size greater than one is used to speed up the alignment algorithm. In the initial pass, an *ith offset* is calculated that identifies the *i*th best alignment between the data sets. In subsequent passes or iterations, the sliding window starts at the *ith offset* – *KN***step_size* (*KN* is an integer < 2^i), searches a reduced *DW* (step 12), and proceeds scanning using a smaller step size (step 13 decrements *i* which is used to set the new step size in step 7). In the final pass, a step size of 1 is used to identify the offset that produces the best alignment between $\boldsymbol{U_x}$ and $\boldsymbol{W}$. **Figure 31** shows the final offset for the current example.

5.4 Implementation of least squares minimization in R

Once data is processed and aligned, power models can be developed and analyzed. In this work, power loads were modeled for subsets of *virgo2* cluster nodes. This section describes how the least squares problems are formulated and implemented in R. **Chapter 6** provides additional details and specific results.

5.4.1 Definition of least squares matrix *A*

After processing /proc variables, samples are stored in the R environment as a set of 10 rectangular matrices, one matrix for each /proc file. In the following discussion, a **bolded** /proc file denotes a matrix that contains processed samples. For instance, **stat** (for a 1-node/1-processor/1-core HPCC computation) looks similar to **Table 20** in the R environment.

Table 20 – The first six columns (target variables) of **stat** in R.

user	*sys*	*sysi*	*iow*	*hirq*	*sirq*	*…*
0	0	800	0	0	0	…
0	0	799	0	0	0	…
0	0	800	0	0	0	…
0	0	800	0	0	0	…
0	0	799	0	0	0	…
39	51	689	2	1	17	…
2	4	793	0	0	0	…
43	730	27	0	0	0	…
387	413	0	0	0	0	…
430	369	0	0	0	0	…
437	364	0	0	0	0	…
427	372	0	0	0	0	…
434	366	0	0	0	0	…
433	367	0	0	0	0	…
432	368	0	0	0	0	…
…	…	…	…	…	…	…

Let **PROC** be the set of /proc matrices in R, **PROC** = {**buddyinfo**, **diskstats**, **interrupts**, **loadavg**, **meminfo**, **net_dev**, **schedstat**, **slabinfo**, **stat**, **vmstat**}. The dimensions of a matrix are denoted $N_{\mathbf{proc\_x}} \times n_{\mathbf{proc\_x}}$, where **proc_x** $\in$ **PROC**, $N_{\mathbf{proc\_x}}$ is the number of **proc_x** rows (samples) and $n_{\mathbf{proc\_x}}$ denotes the number of columns (target variables). For the example shown in **Table 20**, $N_{\mathbf{stat}}$ is 39,400 and $n_{\mathbf{stat}} = 6$. By necessity, $N_{\mathbf{proc\_x}} = N_{\mathbf{proc\_y}}$ for **proc_x**, **proc_ y** $\in$ **PROC**, **proc_x** $\neq$ **proc_ y**. That is, the number of samples across all proc matrices must be equal. For cases where this is not the case, all files are truncated to match the number of samples

of the smallest data set, which usually is not more than one buffer size (100 samples in the current work). This means the number of rows can be uniformly represented using N.

In general **proc_x** is a matrix of the form [$\boldsymbol{proc\_x}^1$ $\boldsymbol{proc\_x}^2$ … $\boldsymbol{proc\_x}^n$], where $\boldsymbol{proc\_x}^i$ denotes the ith column vector or variable of **proc_x** ($i = 1 \ldots n_{\boldsymbol{proc\_x}}$). Referencing **Table 20**, **stat** = [$\boldsymbol{stat}^1$ $\boldsymbol{stat}^2$ … $\boldsymbol{stat}^6$ …] = [***user sys sysi iow hirq sirq*** …] where ***user***, ***sys***, ***sysi***, ***iow***, ***hirq***, and ***sirq*** are column vectors of utilization per sample values. Throughout this document, **bolded** variable names reference R column vectors whereas non-bolded variables reference the variable in the context of /proc files.

A system activity matrix $\boldsymbol{A}$ can be formed from any subset of columns of **PROC** matrices. In the worst case the system activity matrix consists of the entire set of *target* variables which results in an $\boldsymbol{N} \times 162$. For this case, define

$$\boldsymbol{A}_{\mathbf{PROC}} = [\boldsymbol{stat}^1\ \boldsymbol{stat}^2 \ldots \boldsymbol{stat}^{15} \ldots \boldsymbol{schedstat}^1\ \boldsymbol{schedstat}^2 \ldots \boldsymbol{schedstat}^{24} \ldots \boldsymbol{vmstat}^{18}]$$

Then any matrix $\boldsymbol{A}$ consists of a subset of columns of $\boldsymbol{A}_{\mathbf{PROC}}$, denoted $\boldsymbol{A} \subseteq \boldsymbol{A}_{\mathbf{PROC}}$.

5.4.2 Definition of least squares column vector $\boldsymbol{b}_M$

Let $\boldsymbol{A}_M \subseteq \boldsymbol{A}_{\mathbf{PROC}}$ denote the system activity matrix (as described above) of a metered compute node, M and let and $\boldsymbol{b}_M$ denote the corresponding node's *active power* consumption vector, where M = c0-11 or c0-12. The vector $\boldsymbol{b}_M$ is simply an $N \times 1$ column vector that comprises *Watts_delta* values. At this point, two types of power models can be developed as outlined in the next two sections based on system activity of either a non-metered, $\boldsymbol{A}_C$, or metered, $\boldsymbol{A}_M$, compute node together with the power load vector of a metered node, $\boldsymbol{b}_M$. In the following discussions, a power model is defined by its coefficient vector, $\boldsymbol{x}$. Each type of model is evaluated in **Chapter 6**.

5.4.3 Modeling power load vectors of non-metered compute nodes, *C*

This section describes estimating the power load of a non-metered compute, C, node given a set of coefficients, $\boldsymbol{x_M}$. To find $\boldsymbol{x_M}$ (the vector of coefficients that identifies the variables that contribute to total power consumption) we first solve the following equation:

Eq. 5.1 $$\boldsymbol{x}_M = (\boldsymbol{A}_M{}^{\mathrm{T}}\boldsymbol{A}_M)^{-1}(\boldsymbol{A}_M{}^{\mathrm{T}})\,\boldsymbol{b}_M$$

Where $\boldsymbol{A}_M$ and $\boldsymbol{b}_M$ are defined as in **Section 5.4.2**. In R, **Eq. 5.1** is implemented using the `solve`() function as follows: $\boldsymbol{x}_M$ = `solve`($\boldsymbol{A}_M{}^{\mathrm{T}}{*}\boldsymbol{A}_M$,$\boldsymbol{A}_M{}^{\mathrm{T}}{*}\boldsymbol{b}_M$). In this case, $\boldsymbol{x_M}$ is the vector of coefficients that define the model for compute node *M*. **Eq. 5.1** implements the training phase of the model (Pathak et al., 2011). In the current work, the variables that make up $\boldsymbol{A}_M$ and $\boldsymbol{x}_{MM}$ are referred to as the training variables for the model.

Next, the power load ($\boldsymbol{b}_C$) of a non-metered compute node, *C*, where *C* = 1 … *max_nodes*, is modeled using **Eq. 5.2**:

Eq. 5.2 $$\boldsymbol{b}_C = \boldsymbol{A}_C\,\boldsymbol{x}_M$$

Note that the first modeling approach, defined by application of **Eq. 5.1** and **5.2**, makes two assumptions: 1) the variables that track system utilization on a metered node, *M*, are linked or correlated with power consumption on the metered node (**Eq. 5.1**); and 2) the coefficients trained on the metered node can be applied to system utilization of the same variables on non-metered compute nodes to model power loads (**Eq. 5.2**).

The first modeling approach essentially trains models on metered compute nodes and applies the resulting coefficients to non-metered nodes to provide non-metered node power load estimates. This approach makes sense for a homogeneous cluster such as *virgo2* where one would expect system utilization that describes power consumption of each modeled compute node is characterized by the same variables that characterize power loads for metered nodes. In

these cases, system activity matrices for both metered and non-metered nodes consist of the same variables. In other words, for any two compute nodes M and C, there is a one-to-one correspondence between the columns of matrix $\boldsymbol{A}_M$ and the columns of matrix $\boldsymbol{A}_C$, where $M \neq C$, and M corresponds to the index of a metered compute node, and C is the index of a non-metered node.

Three questions arise that are addressed in **Chapter 6**:

1) Which variables do we select as it relates to **Eq. 5.1**?
2) How accurate are those variables in describing power loads of the metered nodes?
3) How accurate are those variables in describing power loads of non-metered nodes (**Eq 5.2)**?

In summary, the approach discussed in this section is useful for building general models based on a representative set of variables that describe both system utilization and power consumption across an entire homogeneous cluster. Applying **Eq. 5.1** and **5.2** correspond to utilization-based power modeling as described in (Pathak et al., 2011). **Eq. 5.1** corresponds to the "*training phase*" because the coefficients are calculated or "*trained*" based on a utilization sample, of system activity. **Eq. 5.2** corresponds to the "estimation or modeling phase", where the model coefficients are used to estimate power load based on system activity on non-metered nodes. Note that these coefficients can be applied either "offline" or at "run-time" on any node. Offline analysis is conducted throughout **Chapter 6**. Finally, **Section 6.12** applies the modeling approach described in this section to estimate energy consumption of the entire *virgo2* cluster. [9]

[9] Note that there is an implied feedback process that can be applied on metered nodes where the first calculation of $\boldsymbol{x}_M$ is initially trained based on a sufficiently sized sample of system activity $\boldsymbol{A}_M$. An online program can then simultaneously measure power load and run-time utilization, buffer sample sets, synchronize if necessary, then compare both actual and predicted power load and make adjustments to $\boldsymbol{x}_M$ as necessary. The coefficients, x_M, can in turn be distributed to compute nodes that apply the coefficients to estimate their power loads. This is a topic that can be addressed by future work.

5.4.4 Training models on non-metered compute nodes

Let $\boldsymbol{A}_C$ and $\boldsymbol{b}_M$ be defined as specified in the previous section(s). The following equation (**Eq. 5.3**) can be used to "fit" a non-metered compute node's system activity matrix to a metered-node's power load vector. (This is essentially the same as **Eq. 5.1**, with the exception that $\boldsymbol{A}_C$ is used here instead of $\boldsymbol{A}_M$). In the current work, the model variables in this case are referred to as "*fitted variables*" since the system activity measured by the variables on one node are fitted to power load of a different node. **Eq. 5.3** shows a vector of coefficients ($\boldsymbol{x}_C$) is calculated for each non-metered compute node based on a metered computed node's power load vector.

Eq. 5.3 $$\boldsymbol{x}_C = (\boldsymbol{A}_C{}^{\mathrm{T}} \boldsymbol{A}_C)^{-1} (\boldsymbol{A}_C{}^{\mathrm{T}}) \boldsymbol{b}_M$$

Eq. 5.4 $$\boldsymbol{b}_C = \boldsymbol{A}_C \boldsymbol{x}_C$$

Another way to think of **Eq. 5.3** is that power consumption of a metered node, $\boldsymbol{b}_M$, is used as a proxy for power consumption of a non-metered node. Applying the coefficients ($\boldsymbol{x}_C$) to **Eq. 5.4** then provides a second estimate (b_C) for that node's power consumption that can be applied to arbitrary system activity matrices for that node or run-time utilization. The underlying premise behind **Eq. 5.3** is that each *virgo2* compute node is homogeneous and thus should consume approximately the same amount of power during computational runs. In **Section 6.13.2** the modeling approach described in this section is used to evaluate the degree of homogeneity in HPCC computations and power consumption across compute nodes.

5.5 Power modeling algorithms

Two approaches were used to select variables to train power models in the current work with the goal of identifying the variables that both measure utilization of the main system level power consumers and can be applied as shown in **Eq. 5.1**. The two variable selection approaches

are 1) manually select variables based on research combined with trial-and-error; and 2) automatically select variables by applying rules and statistical calculations. Essentially, the second approach is a materialization of the manual approach and attempts to automate the task of selecting variables and solving for coefficients. This section describes the base algorithm and two variants of this algorithm.

5.5.1 Algorithm 1 – Variable enumeration

Let $\boldsymbol{V}_{\mathbf{PROC}}$ denote the set of column vectors or *target* variables that remain after sampling, processing and aligning /proc files with Watt meter files. The algorithm proceeds as follows:

1. Start with an empty matrix $\boldsymbol{A}$.
2. Let $\boldsymbol{b}$ represent *Watts_delta* for the modeled node.
3. Find the next column vector $\boldsymbol{v} \in \boldsymbol{V}_{\mathbf{PROC}}$ such that when added to $\boldsymbol{A}$ ($\boldsymbol{A} = \boldsymbol{A} \cup \boldsymbol{v}$) minimizes the sum of squared residuals (see **Eq. 5.1**).
4. Remove $\boldsymbol{v}$ from $\boldsymbol{V}_{\mathbf{PROC}}$ if one of these conditions is met:
   - $\boldsymbol{v}$ is the vector that minimizes the sum of the squared residuals.
   - $\boldsymbol{v}$ is a vector that makes $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$ singular (making the least squares formulation unsolvable).
5. Remove $\boldsymbol{v}$ from $\boldsymbol{A}$ if one of these conditions is met:
   - $\boldsymbol{v}$ is a vector that makes $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$ singular (making the least squares formulation unsolvable).
6. If $\boldsymbol{V}_{\mathbf{PROC}}$ is an empty matrix stop, otherwise repeat step 3.

In summary, *Algorithm 1* begins by finding the vector $\boldsymbol{v} \in \boldsymbol{V}_{\mathbf{PROC}}$ that accounts for the greatest amount of variance in the data (power samples) compared to all other vectors in $\boldsymbol{V}_{\mathbf{PROC}}$. Subsequent iterations find the next vector in $\boldsymbol{V}_{\mathbf{PROC}}$ (the remaining set of vectors) that when

combined with the running set of selected vectors ($\boldsymbol{A}$) together account for the greatest amount of remaining variability. When the algorithm terminates, it has found as many /proc utilization counters as possible that account for the greatest variability in the modeled data (power measurements). In this algorithm, $\boldsymbol{A}$ can consist of either metered or non-metered compute node system activity. On the other hand, $\boldsymbol{b}$ can only be a metered compute node's power samples.

Algorithms 2 and *3* are variants of *Algorithm 1* that simply add constraints to the base algorithm:

- *Algorithm 2* (listed below) works exactly the same as *Algorithm 1* with a slight twist. For each iteration of the main loop, the running coefficient vector $\boldsymbol{x}$ is constrained to be greater than $\mathbf{0}$. The vector $\boldsymbol{x}$ is greater than zero iff $x_i > 0$ for all $x_i \in \boldsymbol{x}$, i = 1…num_elements($\boldsymbol{x}$), where num_elements($\boldsymbol{x}$) is the number of elements in $\boldsymbol{x}$.
- *Algorithm 3* (listed below) adds the following constraint to *Algorithm 2*: Add $\boldsymbol{v}$ to $\boldsymbol{A}$ iff the correlation between $\boldsymbol{b}$ and $\boldsymbol{v}$, corr($\boldsymbol{b}$,$\boldsymbol{v}$), is greater than the correlation threshold. In the current work, the global variable CORR specifies the correlation threshold, which by default is 0.25. Also note, an R script that implements *Algortithm 3* works as follows: if a negative value of CORR is specified, then the algorithm will add $\boldsymbol{v}$ to $\mathbf{A}$ iff corr($\boldsymbol{b}$,$\boldsymbol{v}$) ≤ CORR or CORR ≥ corr($\boldsymbol{b}$,$\boldsymbol{v}$). If a positive value of CORR is specified, the algorithm will add $\boldsymbol{v}$ to $\boldsymbol{A}$ iff CORR ≥ corr($\boldsymbol{b}$,$\boldsymbol{v}$).

5.5.2 Algorithm 2 – Variable enumeration with $x > 0$

Let $\boldsymbol{V}_{\mathbf{PROC}}$ denote the set of column vectors or *target* variables that remain after sampling, processing and aligning /proc files with Watt meter files. The algorithm proceeds as follows:

1. Start with an empty matrix $\boldsymbol{A}$.
2. Let $\boldsymbol{b}$ represent *Watts_delta* for the modeled node.

3. Find the next column vector $\boldsymbol{v} \in \boldsymbol{V}_{\mathbf{PROC}}$ such that when added to $\boldsymbol{A}$ ($\boldsymbol{A} = \boldsymbol{A} \cup \boldsymbol{v}$) minimizes the sum of squared residuals (see **Eq. 5.1**) and $\boldsymbol{x} > 0$.
4. Remove $\boldsymbol{v}$ from $\boldsymbol{V}_{\mathbf{PROC}}$ if one of these conditions is met:
   - $\boldsymbol{v}$ is the vector that minimizes the sum of the squared residuals.
   - $\boldsymbol{v}$ is a vector that makes $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$ singular (making the least squares formulation unsolvable).
   - $\boldsymbol{v}$ is a vector such that at least one element of $\boldsymbol{x}$ is less than or equal to 0.
5. Remove $\boldsymbol{v}$ from $\boldsymbol{A}$ if one of these conditions is met:
   - $\boldsymbol{v}$ is a vector that makes $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$ singular (making the least squares formulation unsolvable).
   - $\boldsymbol{v}$ is a vector such that at least one element of $\boldsymbol{x}$ is less than or equal to 0.
6. If $\boldsymbol{V}_{\mathbf{PROC}}$ is an empty matrix stop, otherwise repeat step 3.

5.5.3 Algorithm 3 – Variable enumeration with $x > 0$ and correlation $(b, v) >$ CORR

Let $\boldsymbol{V}_{\mathbf{PROC}}$ denote the set of column vectors or *target* variables that remain after sampling, processing and aligning /proc files with Watt meter files. The algorithm proceeds as follows:

1. Start with an empty matrix $\boldsymbol{A}$.
2. Let $\boldsymbol{b}$ represent *Watts_delta* for the modeled node.
3. Find the next column vector $\boldsymbol{v} \in \boldsymbol{V}_{\mathbf{PROC}}$ such that when added to $\boldsymbol{A}$ ($\boldsymbol{A} = \boldsymbol{A} \cup \boldsymbol{v}$) minimizes the sum of squared residuals (see **Eq. 5.1**) and $\boldsymbol{x} > 0$ and the correlation between $\boldsymbol{b}$ and $\boldsymbol{v}$ satisfies the correlation constraint (CORR = 0.25).
4. Remove $\boldsymbol{v}$ from $\boldsymbol{V}_{\mathbf{PROC}}$ if one of these conditions is met:
   - $\boldsymbol{v}$ is the vector that minimizes the sum of the squared residuals.
   - $\boldsymbol{v}$ is a vector that makes $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$ singular (making the least squares formulation unsolvable).
   - $\boldsymbol{v}$ is a vector such that at least one element of $\boldsymbol{x}$ is less than or equal to 0.

- $\boldsymbol{v}$ is a vector such that correlation between $\boldsymbol{b}$ and $\boldsymbol{V}^{\boldsymbol{K}}$ < CORR

5. Remove $\boldsymbol{v}$ from $\boldsymbol{A}$ if one of these conditions is met:
   - $\boldsymbol{v}$ is a vector that makes $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$ singular (making the least squares formulation unsolvable).
   - $\boldsymbol{v}$ is a vector such that at least one element of $\boldsymbol{x}$ is less than or equal to 0.
6. If $\boldsymbol{V}_{\mathbf{PROC}}$ is an empty matrix stop, otherwise repeat step 3.

5.5.4 Discussion

The primary differences between *Algorithm 1* and *Algorithm 2* are as follow. The former generally produces more accurate models (i.e. the final sum of squared residuals tends to be smaller than corresponding values for *Algorithm 1*). However, the algorithm tends to produces negative coefficients (as discussed in **Chapter 6**). This implies that variables are contributing negative Watts or power consumption to total Watts. For this reason *Algorithm 2* adds the constraint that $\boldsymbol{x}$ be greater than 0. This makes analysis of each variable's contribution to total power possible (**Section 6.13**). The tradeoff, however, is that *Algorithm 2* results in larger sum of squared residuals. *Algorithm 3* is an enhancement over *Algorithm 2* that eliminates variables whose standard deviation becomes 0 when Watt window sizes are adjusted and selects variables based on the specified correlation threshold. **Chapter 6** provides results obtained from applying these three algorithms to selecting variables for models.

Note 1: one possible explanation for negative coefficients that result in *Algorithm 1* may be that the algorithm is subtracting out the intersection of two or more variables whose values capture overlapping system behavior. This is explored in a little more detail in **Section 6.13**, with in depth analysis of negative coefficients left for future work.

Note 2: When applying **Eq. 5.1** and **Eq. 5.3** to *Algorithms 1*, *2*, and *3*, the resulting coefficient vectors or activity matrices selected by each algorithm will tend to differ across compute nodes (as shown in **Chapter 6**) even when all nodes are executing the same HPCC

benchmarks. This is in part due to each algorithm selecting the set of variables in an order that results in a minimal sum of least squares residuals. While the approach of these algorithms seems to characterize the uniqueness of a compute node's system utilization and corresponding power consumption when calculating the HPCC benchmarks, it complicates identifying a common set of variables that describes power consumption of an entire cluster. This complication is addressed in **Chapter 6** by applying *Algorithms 1*, *2*, and *3* in combination with factor analysis (factor analysis is briefly covered in **Section 5.6.2**) to identify a broad set of variables that link system utilization with power consumption. The chosen variables are then further reduced to a general set of variables common across all compute nodes for full cluster power modeling.

5.6 R Power Modeling Tools

While power modeling and analysis in R is facilitated with scripts, functions, libraries, packages, and graphics, there are some parts of an analysis that are tedious and time consuming. For example, **Figure 32** shows a Watt meter sample that includes two HPCC computations launched by OpenMPI. The first set of samples shows HPCC with processor affinity (the HPCC process is bound to the selected core by MPI) and the second shows HPCC without affinity. After this data is aligned with /proc samples, if one wants to build a power model based on a specific Watt window (i.e. to develop a power model of HPCC with affinity only) to do so requires calculating the lower and upper bounds of the ranges in question (which requires specifying the Watt window and sliding windows, refer to **Figures 30** and **31**), then passing these parameters to R functions that perform calculations. An R function prototype that implements least squares regression is shown in **Figure 33**. Frequently calling functions with different parameters either via the R command line or by editing R scripts has the potential to be time consuming and tedious, especially when many sections of the HPCC benchmark are analyzed with different Watt windows. To facilitate interactive and dynamic modeling and analysis, this work makes use of the *rpanel* package (rpanel, 2007) described next.

5.6.1 The *rpanel* package

Rpanel is an R package that provides a means of building simple interactive controls for functions (rpanel-2007). This package makes it possible to not only implement controls for functions, but to apply those controls to interactively model power consumption and dynamically display graphical results. In this work, rpanel is used to implement three types of graphical user interfaces: global options, Watt window controls, and compute node power modeling control.

5.6.2 Global Options rpanel

Figure 34 shows an rpanel that provides global options. This panel enables fine grained loading of proc utilization data and Watt meter data (columns labeled Node Data). It also provides options to align data (Model Options) and select a Watt baseline (for computations involving more than one metered node). Once data is loaded into memory and aligned, "Watt Window" and compute node "Proc Power Model" panels can be opened (via the buttons shown on the far right of the image).

5.6.3 Watt Window rpanel

Figure 35 shows a Watt window rpanel. This panel is used to set global Watt window parameters that can be applied to more than one compute node's power modeling panel. The main functions of this window are to resize a global Watt window. Resizing the Watt window is accomplished using the sliders on the far right of the figure. Lower and Upper bounds can be set in three levels of granularity. Data can also be aligned relative to resized windows from here. This is useful for situations where data is not aligned properly across an entire data set. The Watt window can be adjusted to the interval of interest and aligned with /proc data.

5.6.4 Compute node power modeling rpanel

Figure 36 shows a compute node "Proc Power Model" rpanel. This panel enables both local and global analysis. That is, Watt windows can be adjusted in a similar manner as specified above, where the adjusted Watt window will only apply to this node's power model. Optionally, an option can be set that reads global Watt window settings and builds models based on those settings.

This panel allows four types of model building:

- Custom models are calculated with the option "Find x – Selected". First, variables of interest are selected by check marking boxes in columns labeled *stat*, *interrupts*, etc. Once variables of interest are selected, the "Find x –Selected" option will calculate the least squares coefficients that correspond to the selected variables and resulting activity matrix.
- The options labeled "Find x – Alg 1" through "Find x – Alg 3" implement the algorithms specified in **Section 5.4**. The following procedure illustrates how to build models based on this algorithm for a specified Watt window:
    1. By default, all variables are considered. To omit variables from consideration either select one of the proc file checkboxes in the first column (*stat*, *interrupts*, etc) or checkmark a variable in the proc columns.
    2. Run the algorithm by selecting one of the "Find – x Alg" checkboxes.
- Models can be applied to dynamically adjusted Watt windows
- A range of output options can be selected to provide graphical and tabular results

Chapter 6 provides results obtained from applying the R modeling tool to various HPCC benchmark calculations.

5.6 Coefficient of Determination and Factor Analysis

This section describes two tools used in the variable selection process and used to evaluate power models. These are the Coefficient of Determination and Factor Analysis. The former was coded into R scripts and the latter is available as a built-in function, `factanal`.

5.6.1 Coefficient of Determination, (CoefD or R^2)

The coefficient of determination (CoefD) provides a measure of the degree to which a model predicts future outcomes (CoefD, 2009). In a regression equation, CoefD provides a measure of the degree of variation in the dependent variable that is explained or accounted for by the independent variable. In the context of the current work, CoefD is the proportion of variance in the power load samples (the data set) that is accounted for by the predicted power per sample. The value of CoefD is calculated by taking the ratio of the variation in the independent variable (the explained variation) to the variation in the dependent variable or total variation (CoefD, 2012). In the current work, the following equation is used:

Eq. 5.4 $$CoefD = 1 - (SSE/SST)$$

Where

- SSE = the sum of squared deviations of the errors in observed power loads from predicted power loads (Knight, 1980) (CoefD, 2004). SSE is also referred to as the sum of the squared residuals. SSE is calculated using **Eq. 5.5** below.
- SST = the sum of squared deviations of the observed power loads from the mean of power loads (CoefD-2004). SST is calculated using **Eq. 5.6** below.

Eq. 5.5 $$SSE = \sum_{i=1}^{N} (b_i - b)_i^{'}$$

Eq. 5.6 $$SST = \sum_{i=1}^{N} \left(b_i - b_{avg} \right)$$

Where,

- b_i = *i*th power load sample
- b_{avg} = mean of power load samples
- b_i' = *i*th predicted power load sample

In R, the following code calculates CoefD, where, M is the vector of predicted power loads which corresponds to $\boldsymbol{b}_C$ in **Eq. 5.2** and ***b'*** in **Eq. 5.5**:

```
WR2_sum = sum((b - ave(b))^2)
R2_sum  = sum((b - M)^2)
CoefD_R2 = 1 - R2_sum / WR2_sum
```

5.6.2 Factor analysis, FA

Factor analysis (FA) is a technique that takes many measurements and resolves them "into distinct patterns of occurrence" (Rummel, 2012). Regularity in data is identified that can be described "as a pattern of variation" (Rummel, 2012). FA effectively identifies patterns of relationship among variables. It is also a data reduction technique where the aim is "to find the fewest common dimensions among a set of variables" (Feser, 2005).

In R, maximum-likelihood factor analysis (Steiger, 2009) (Miles, 2005) (Kabacoff, 2012) is performed using the function `factanal` (R, factanal). In the current work, factor analysis is applied to power load vectors (samples) and system utilization vectors (samples) of *target* variables in order to identify groups of related variables or clusters whose patterns of variation describe the utilization and power loads of the same system component. To the best of my knowledge, factor analysis as applied in the context of the current work for power modeling has not been conducted. There is research however, that has applied factor analysis in the context of analyzing performance data (Delic et al., 1996) (Jain, 1991). **Section 6.9** provides additional

information on the use of factor analysis in the current work. The section applies exploratory factor analysis (EFA) to utilization variables and power load vectors to gain intuition on the structure of the variables with respect to "unobserved" power consumption of internal components: no attempt is made to identify variables based on prior theory and test whether they load as specified by theory (Tucker and MacCallum, 1993) (Garson, 2012).

5.7 Chapter Summary

This chapter discusses processing steps applied to power meter and /proc samples collected during HPCC trials. As it relates to /proc files, data samples need to reflect system activity or utilization per sample, thus, cumulative counters are first processed to reflect such activity. For instance, number of transmitted packets is a cumulative count per sample that is converted to number of transmitted packets per sample. Next constant counters are removed from analysis because they do not provide meaningful utilization information. The chapter then describes how least squares minimization problems are formulated with /proc and Watt meter samples after data is aligned using an automated alignment algorithm. Two general modeling approaches are specified and three variable selection algorithms are described which automate the selection of model variables by enumerating through all variables in a search set. While the three algorithms facilitate variable selection, R GUIs facilitate analysis and model generation. The many advantages of R show that it is suitable for this type of work. The chapter concludes with a description of the two primary tools used in **Chapter 6** to develop system level and full cluster power models: the coefficient of determination and factor analysis.

Chapter 6: Results

Recall the /proc file system contains system performance and utilization information that is tracked by the Linux kernel. Some files track utilization of hardware components and these components in turn consume energy by drawing power to operate. Thus, utilization counters found in the /proc file system provide a measurement of hardware utilization that can be linked to power consumption. This concept is described by (Pathak et al., 2011) as the fundamental assumption of utilization-based power modeling: i.e. utilization of a hardware component corresponds to a certain power load and a change in utilization (triggered by some event) results in a change in power load of the device. This chapter analyzes the link between /proc files that track system hardware utilization and power consumption. Least squares coefficients that define power models are derived for individual compute nodes and for an entire cluster. This chapter shows different sets of coefficients emerge that describe power consumption of a compute node when the objective is to minimize the sum of the squared residuals using an automated approach. The set of selected model variables and their values (coefficients) depend on a variety of factors including the number cores involved in an HPCC computation and the number of samples analyzed. Also, there is a tradeoff between model accuracy (as measured by the sum of the squared residuals) and the model's explanatory power (number of components accounted for). Throughout this chapter, power models and results are evaluated using a variety of tools including:

- Pearson's correlation coefficient, Corr (R, cor).
- The average of the sum of squared residuals, R2_Sum_avg.
- The coefficient of determination, CoefD.
- Factor analysis (R, factanal).

As with previous chapters, all tables and figures referenced in this chapter are found in **Appendix FT6**. Those appearing in this chapter are reprinted from the appendix.

6.1 Node/Processor/Core Configurations

In the current work, all power models were developed based on data collected during HPCC trials. In most cases, each trial consisted of two HPCC computations executed on a subset of *virgo2* cores (160 in total), the first with OpenMPI processor affinity (described below) and the second without. Each computation was separated by a sufficient amount of idle time to ensure compute node power loads returned to their lowest levels and remained stable. An example of an HPCC trial is shown in **Figure 37**, reprinted in **Section 6.3**).

Table 21 – HPCC benchmark computation configurations.

Ref#	Configuration	# Compute Nodes	# Processors	# Cores	MPI Affinity	N
1	*n*1*p*1*c*1*a*1	1	1	1	1	31,000
2	*n*1*p*1*c*1*a*0	1	1	1	0	31,000
3	*n*1*p*1*s*0*c*4*a*1	1	1	4	1	31,000
4	*n*1*p*1*s*1*c*4*a*1	1	1	4	1	31,000
5	*n*1*p*2*c*4*a*1	1	2	4	1	31,000
6	*n*1*pX*c4*a*0	1	(1-2)	4	0	31,000
7	*n*1*p*2*c*8*a*1	1	2	8	1	31,000
8	*n*1*p*2*c*8*a*0	1	2	8	0	31,000
9	*n*2*p*4*c*16*a*1	2	4	16	1	43,000
10	*n*2*p*4*c*16*a*0	2	4	16	0	43,000
11	*n*4*p*8*c*32*a*1	4	8	32	1	61,000
12	*n*4*p*8*c*32*a*0	4	8	32	0	61,000
13	*n*20*p*40*c*160*a*1	20	40	160	1	90,000
14	*n*20*p*40*c*160*a*0	20	40	160	0	90,000

Table 21 uniquely identifies (Ref#) the HPCC computations that make up a trial and their corresponding configurations. Each *trial* is indicated with alternating row shades (thus there are six trials shown in the table). The column labeled "Configuration" uses a shorthand notation to represent the number of compute nodes, *n*, the number of processors, *p*, and the number of *c* cores involved in each benchmark calculation. Hardware configurations are referenced with an *n*/*p*/*c* triple. Other characteristics applied to a configuration are denoted by appending a character

that represents that characteristic followed by a value for that characteristic. For instance, in **Table 21**, appending *a*1 means OpenMPI launches HPCC processes with processor affinity and *a*0 means OpenMPI launches HPCC processes without processor affinity (the default). MPI processor affinity binds a process to a core for the duration of an HPCC computation. Launching HPCC without affinity means OpenMPI is free to redistribute processes among cores in an attempt to load balance (OpenMPI, 2012).

In some cases, more than one trial is executed for a particular configuration. Multiple trials are identified by appending "*tK*" to the configuration, where *K* denotes the *K*th trial. For instance, in later sections we will see three trials corresponding to hardware configuration *n*1*p*2*c*8: *n*1*p*2*c*8*aXt*1, *n*1*p*2*c*8*aXt*2, and *n*1*p*2*c*8*aXt*3. Note the use of *aX*: this is used to reference both HPCC calculations for the trial, whereas the notations *n*1*p*2*c*8*a*1*t*1 and *n*1*p*2*c*8*a*0*t*1 reference the particular HPCC computation within the trial.

Also note in **Table 21** that trial *n*1*pXc*4 consists of four successive HPCC computations. This is because two types of communications related mappings are possible when allocating four processes on a single node and each type consists of two cases. First, all four processes can be assigned and bound to a single processor (with the first case assigning all processes to the first processor and the second case assigning all processes to the second processor). These instances involve core-to-core (intra-processor) communications and do not involve processor-to-processor communications. Second, the four processes can be divided among the eight cores of the two processors. The type of communication involves both core-to-core and processor-to-processor (intra-node) communications. The two cases are launching the four processes with affinity or without.

In the current work, when four processes are bound to a single CPU (Ref# 3 and 4 for instance) an additional *slot* identifier is used. This indicates which processor HPCC processes are bound to with *s*0 indicating a binding of four processes to CPU0 and *s*1 indicating a binding of four processes to CPU1. Slot binding is accomplished in OpenMPI with *rankfiles* (OpenMPI, 2012). Rankfiles contain user specified mappings of processes to cores and the files can specify

which slot to use in a mapping. In the current work, *rankfiles* were used to compute the HPCC benchmarks on each single processor to isolate both computation and communications (only intra-processor communication is involved between HPCC processes). Ref# 5 shows a situation where two processes are bound to each CPU on a single node, in which case inter-processor communication is involved. Once again *rankfiles* were used to force this mapping in the current work. If the mapping is not done using *rankfiles*, there is the risk that OpenMPI will unevenly allocate processes. Ref# 6 is the four core no-affinity case (no rankfiles used in this case and OpenMPI launched HPCC without specifying affinity). For this case, *pX* indicates that OpenMPI may assign all processes to one processor or among the processors depending on its internal algorithm.

For each configuration shown in **Table 21**, the optimal problem size was first determined prior to data collection. That is, the HPCC benchmarks were computed for various problem sizes until the value of *N* that resulted in peak HPL Flops for the configuration was found. There are two exceptions, Ref#'s 13 and 14, where time constraints limited the problem size to $N = 90{,}000$. After finding *N*, an official trial was conducted to collect /proc utilization and power measurements. For single node trials (*n*1), power measurements were made on a single compute node, c0-11. Power measurements were made on two compute nodes (c0-11 and c0-12) for all other trials (i.e. all configurations that involved two or more compute nodes, $n > 1$). To identify the metered compute node for each trial, either *w*11 (c0-11) or *w*12 (c0-12) is appended to the appropriate configuration string.

What is changing between configurations? Recall, each configuration in **Table 21** uses a different number of cores to compute the HPCC benchmarks. As the number of cores increase, the level of communications increases. In some cases, an increase in the number of cores also indicates that a trial utilizes different system components. For instance, *n*1*p*1***c*1***a*1 involves only a single core and thus no form of communication is needed. Increasing from one to four cores and binding all processes to a single processor, as in *n*1*p*1*s*0*c*4*a*1 and *n*1*p*1*s*1*c*4*a*1, involves intra-processor communication because the four cores need to communicate results. Other

configurations such as *n*1*p*2*c*4*a*1 and *n*1*p*2*c*8*a*1 involve both intra-processor and inter-processor communication on a single node. Data has to be passed not only between processor cores but also between cores on different processors on the same node. Also note that *n*1*p*2*c*8*a*1 involves a higher degree of communications than *n*1*p*2*c*4*a*1: that is, in the former, eight cores need to communicate versus four in the latter. Finally, *n*2*p*4*c*16, *n*4*p*8*c*32, and *n*20*p*40*c*160 involve core-to-core, processor-to-processor, and network communications with the level of network communications increasing with the number of nodes.

In summary, as it relates to **Table 21**, memory utilization and network communications increase for the following sequences {7, 9, 11, 13} and {8, 10, 12, 14}. Inter-processor communication "increases" for the single node (intra-node) sequences {3, 8} and {4, 8}. Intra-processor communication, or communication between cores on a processor, "increases" for sequences {2, 3} and {2, 4}. In general, as the number of nodes, *n*, increase the level of network communications increases. Increasing the number of cores, *c*, from 1 to 4 (for sequences {2, 3} and {2, 4}) increases intra-processor communication, with $c = 8$ increasing both inter-core and inter-processor communication and values of *c* beyond 8 adding network communications to the mix. Finally, as *n*, *p*, and *c* increase, so does system and MPI communications overhead.

6.2 System level power models

System level power models are the basis of analysis presented in this chapter. This section provides background related to developing and evaluating these models. Note in **Table 21,** ten /proc files are sampled for each of the *n* compute nodes associated with the configuration. Recall, for $n = 1$, there is only one metered compute node (c0-11 or *w*11) and for $n \geq 2$, there are exactly two metered compute nodes (c0-11 and c0-12). Throughout **Sections 6.3** through **6.9** system level utilization-based power models are based on data from collected from the metered nodes. **Section 6.13** explores modeling power loads for non-metered compute nodes with the power samples of the metered nodes.

There are three steps for developing system level models. First, a set of /proc variables is selected. Next, given each metered node's power load vector (or column vector of power samples) and system activity matrix (**Section 5.4** defines system activity matrix), a least squares minimization problem is formulated as shown in **Eq. 5.1** (reprinted from **Chapter 5**). Third, the least squares problem is solved resulting in a column vector of coefficients, $\boldsymbol{x_M}$, that correspond to the values of the selected variables.

Eq. 5.1 $$\boldsymbol{x}_M = (\boldsymbol{A}_M{}^{\mathrm{T}}\boldsymbol{A}_M)^{-1}\,(\boldsymbol{A}_M{}^{\mathrm{T}})\,\boldsymbol{b}_M$$

Where

- $\boldsymbol{x}_M$ is the vector of coefficients for the metered compute node, *M*, where *M* is c0-11 or c0-12.
- $\boldsymbol{A}_M$ is the system activity matrix for compute node *M*.
- $\boldsymbol{b}_M$ is the power load vector in Watts for *M*.

The variables selected for the model are referred to as *selected variables* and the corresponding coefficients, $\boldsymbol{x_M}$, define the system level model. Once known, the coefficients are applied as shown in ***Eq. 5.2*** to produce an estimated power load, $\boldsymbol{b}_C$, which essentially models the power load associated with a compute node's system activity matrix. The values of the estimated power load vector are summed to produce an estimate of total energy, E_C, (in kWs) for the modeled node *C*. Recall, total energy is obtained simply by summing over all values of a power load vector as shown in **Eq. 6.1**, where, $\boldsymbol{b}_k$ represents any power load vector, either measured, $\boldsymbol{b}_M$ or modeled $\boldsymbol{b}_C$. This equation corresponds to **Eq. 2.5** where power samples are represented using vector notation typically applied in least squares minimization formulas.

Eq. 6.1 $$\boldsymbol{E} = \sum_{k=1}^{K} b_k$$

In later sections, the accuracy of a model is evaluated by calculating the percent difference of the error in the energy estimate with respect to total measured energy as shown in **Eq. 6.2**.

Eq. 6.2 $$\textbf{\textit{Model Error}} = \frac{E_M - E_C}{E_M} \times 100$$

That is, once $\boldsymbol{x}_M$ is known, the vector can be applied to the original activity matrix, $\boldsymbol{A}_M$ of the metered node to produce an estimated power load, $\boldsymbol{b}'_M$. Energy for both the measured power load and estimated power load, E_M and E'_M respectively are determined and the trained model is evaluated using **Eq. 6.2** where E_C is replaced by E'_M. This process is applied to evaluating trained models in subsequent sections.

Power loads associated with each variable can be produced by multiplying a model coefficient to the corresponding variable's system activity column vector. This is illustrated in **Eq. 6.3**.

Eq. 6.3 $$\boldsymbol{b}_{C,i} = \boldsymbol{A}_{C,i}\, \boldsymbol{x}_{M,i}$$

Where,

- $\boldsymbol{A}_{C,i}$ = Column vector that contains /proc utilization samples of variable *i* on compute node *C*.
- $\boldsymbol{x}_{M,i}$ = Least squares coefficient of variable *i*.
- $\boldsymbol{b}_{C,i}$ = Column vector that models the power load associated with variable *i* on compute node *C*.

Total energy for variable *i* in kWs, **Eq. 6.4**, is obtained by summing over all values of $\boldsymbol{b}_{C,i}$.

Eq. 6.4 $$\boldsymbol{E}_{C,i} = \sum_{k=1}^{N} A_{C,i,k}\, x_{M,i} = \sum_{k=1}^{N} b_{C,i,k}$$

Where,

- $\boldsymbol{A}_{C,i,k}$ = *k*th /proc sample of variable *i* on compute node *C*.
- $\boldsymbol{b}_{C,i,k}$ = *k*th power estimate of variable *i* on compute node *C*.
- $\boldsymbol{E}_{C,i}$ = estimated energy associated with variable *i* on compute node *C*.

This means total (modeled) energy for node *C* can be calculated in two ways. As shown in **Eq. 6.1** or based on **Eq. 6.4** as follows:

Eq. 6.5 $$\boldsymbol{E}_{C} = \sum_{i=1}^{m} E_{C,i}$$

Where, *m* is the number of selected variables. Note that, in the current work, the *selected variables* are also referred to as *training variables* because the power model is *trained* (that is the coefficients are calculated) with respect to the corresponding node's power consumption and utilization data. Also note, the R GUI discussed in **Chapter 5** makes it possible to both manually and automatically select *training variables* for a model.

6.3 Coefficient generation using contiguous HPCC benchmarks

Recall, **Section 5.5** outlined three algorithms implemented in R that automatically select variables and generate coefficients. This section compares the three algorithms as it relates to selected variables. Results are used to evaluate differences between configurations and to help identify a general set of variables that are later applied to modeling full cluster power consumption. It is worth noting that the three algorithms evaluated here are based on automating

the tasks conducted while manually searching for the set of variables that produce the minimal or near minimal value for R2_sum_avg (the average of the sum of squared residuals). Initial analysis related to the current work involved manual selection of variables based on /proc file documentation and trial-and-error. Results indicated that many unique combinations of variables produce similar and comparable power estimates based on very close values of R2_sum_avg. This necessitated a better approach to selecting combinations of variables that was faster, automated, and accurate. *Algorithm 1* is the result. The other two algorithms are attempts to improve the quality of the variables selected.

Figure 37 shows power consumption on compute node c0-11 for an *n*1*p*1*c*1*aX* trial. Total energy for the trial is shown in kWs. Note that idle periods, times when a compute node is not performing work, correspond to power consumption at or near 0 Watts. Recall, raw power samples for a metered node are "processed" by subtracting out the minimum value of idle power (as explained in **Section 5.2.3**). In the current work, the remaining power is referred to as "*active power*". **Figure 38** shows the result of aligning Watt meter data with /proc/stat.*user* samples. The data offset and the correlation between Watt meter and user data are shown at the bottom of the figure. Recall that Watt meter samples and system utilization samples are collected using asynchronous clocks, i.e. the Watt meter uses its own internal clock which is separate from the compute node's clock. The offset shows by how many samples /proc data needs to be shifted in order to align properly with Watt meter samples. The alignment algorithm is described in **Section 5.3.2**.

After /proc data is aligned with Watt meter samples, the data are filtered by eliminating irrelevant information (recall **Section 5.2** describes the elimination process). As mentioned previously, the specific variables that result after the elimination step are referenced using the term "*surviving variables*". Of these, a set of *target* variables is selected. **Table 22** shows the *target* variables per /proc file for the current trial.

At this point, *Algorithms 1*, *2*, and *3* are applied to the *target* variables. The output of each algorithm is the set of *selected variables* described in **Section 6.2**. **Table 23** (reprinted

below) shows the results of applying the three algorithms to the data for the current trial. In the table,

- The first column (under the *Algorithm* identifier) shows the algorithm's *selected variables* ("*Alg. X. Sel. Vars*") and the second column (labeled "x") shows the corresponding coefficients. Bolded variables indicate selection by two or more of the algorithms.
- "Est. kWs" gives each variable's estimated *active* energy in kilowatt seconds. This value is calculated for each variable using **Eq. 6.2**.
- Total estimated *active* energy for the node in kilowatt seconds is shown in the last row of each section of the table. This value is calculated by summing all energy estimates in that column or equivalently using **Eq. 6.3**.
- "R2_sum_avg" is the average of the sum of the squared residuals. This value is calculated by first summing the squared residuals across all samples (the squared residual is square of the difference between measured power and estimated power) and dividing by the total number of samples. For the data shown in the table, the number of samples is 30,925.
- "CoefD" is the value of the coefficient of determination.
- The last three columns are the result of individually applying least squares minimization to each *selected variable*. "Corr_x0" shows the correlation between the variable's utilization samples and Watt meter samples. "R2_sum_avg_x0" and "CoefD_x0" show values for the average of the sum of squared residuals and coefficient of determination.

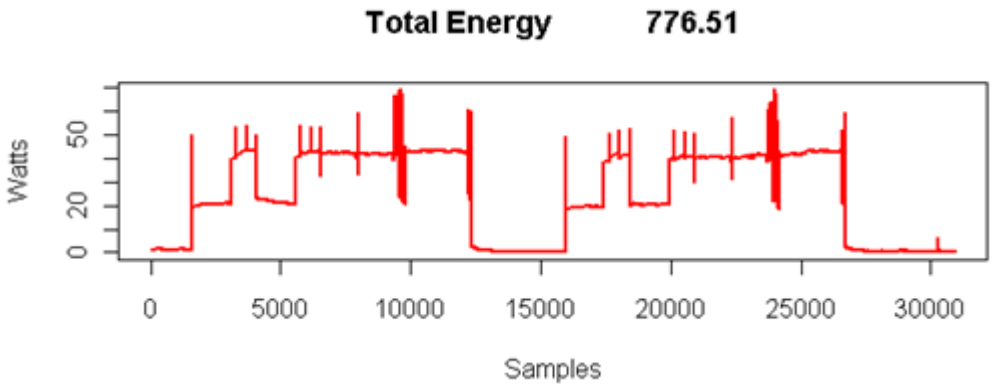


Figure 37 – Watt meter data collected from compute node c0-11.

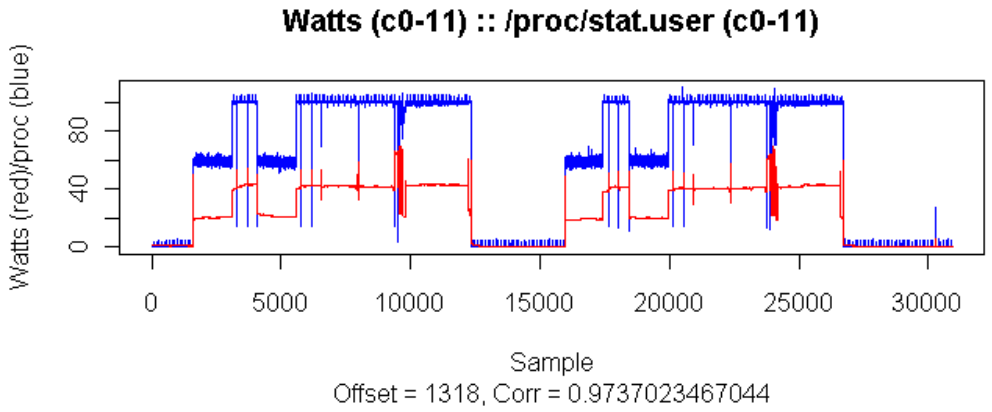


Figure 38 – Aligned Watt meter and /proc samples.

Table 22 – Target variables for *n*1*p*1*c*1*aX* configuration experiment.

/proc file	Target
buddyinfo	16
diskstats	11
interrupts	12
loadavg	5
meminfo	14
net/dev	11
schedstat	24
slabinfo	31
stat	15
vmstat	18
Total	**162**

Note that values in columns R2_sum_avg and CoefD are monotonically decreasing for each set of *selected variables*. This occurs because of the way each algorithm selects variables. That is, the *i*th row for each section (where $i = 1 \ldots number\_selected\_variables$) shows the R2_sum_avg and CoefD values that result when variables from 1 to row *i* are used to calculate coefficients at step *i*. The last row of each section shows the final values (highlighted in light blue) obtained on the last step of the algorithm.

Figure 39 (reprinted below), **Figure 40**, and **Figure 41** provide graphical output for the data in **Table 23**. **Figure 39** shows measured power consumption in red (as measured via the power meter) and estimated power in green. Shown in each sub-title are the number of selected variables, the final values of R2_sum_avg and CoefD (note how these values correspond to the last row of each section of **Table 23**). Total measured (776.51 kWs) and estimated *active* energy (776.59 kWs for *Algorithm 1*, 785.29 kWs for *Algorithm 2* and 779.52 kWs for *Algorithm 3*) are given in parenthesis and separated by a "/" after CoefD. **Figure 40** shows each variable's power contribution to the total power load per sample and **Figure 41** shows the average value of the sum of the squared residuals (R2_sum_avg) as each selected variable is added to the set of selected variables.

Table 23 highlights one of the drawbacks of the first algorithm, namely the selection of variables that have negative coefficients. *Algorithm 1* attempts to find the largest set of variables that best matches measured power consumption. As illustrated in **Section 5.5.1**, the process starts with an empty set of variables. The algorithm then iterates through all variables in the *target set* and selects the one that produces the lowest average error (R2_sum_avg). The selected variable is removed from the *target set* and the process repeats, where the second variable selected is the one when combined with the first (to form a two variable model), produces the lowest average value. The algorithm proceeds until there are no more variables to select. In general, across all trials, *Algorithm 1* consistently selects the set of variables that produces the lowest value of R2_sum_avg.

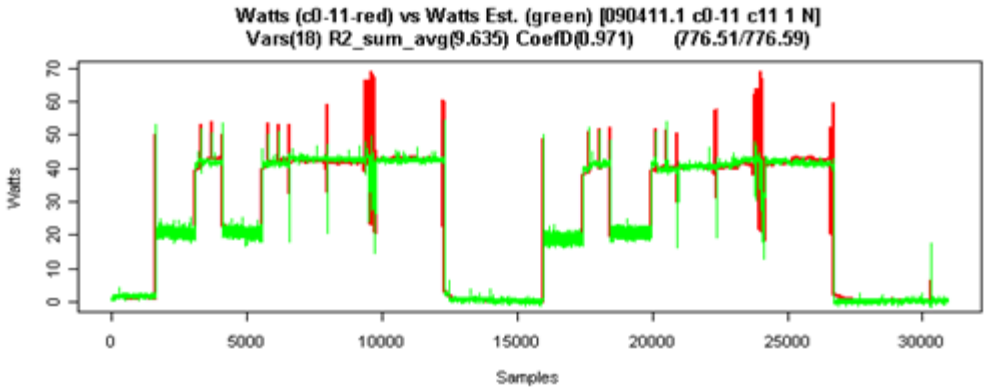
Table 23 – Least Squares results for *Algorithm 1*, *2*, and *3* (*n*1*p*1*c*1 – Contiguous HPCC).

Alg. 1 Sel. Vars	x	Est. kWs	R2_sum_avg	CoefD	R2_sum_avg_x0	Corr_X0	CoefD_X0
user	0.412276	781.93	17.517284	0.948085	17.517284	0.973702	0.948085
pgfault	0.000145	5.92	15.439988	0.954241	957.954352	0.061616	-1.839058
sys	-0.090133	-23.46	12.649698	0.962511	876.374047	-0.107748	-1.597281
active	-0.000002	-220.40	12.188212	0.963878	53.980565	0.918534	0.840020
commitA	-0.000004	-462.05	11.013654	0.967359	56.614126	0.915888	0.832215
pgfree	0.000021	0.88	10.353152	0.969317	966.190532	0.024192	-1.863467
cpu2_6	0.024169	51.03	10.215821	0.969724	434.288791	-0.121113	-0.287087
fil___108_no	0.030048	3.26	10.118123	0.970013	882.308971	0.153976	-1.614870
dma32_7	-0.005405	-132.68	10.037850	0.970251	806.989424	-0.669668	-1.391648
dma32_5	0.004680	141.90	9.976403	0.970433	806.097045	-0.668542	-1.389003
eth0_Tx_B	-0.000003	-1.21	9.926212	0.970582	961.366076	0.000982	-1.849169
int100	0.009640	1.34	9.880291	0.970718	948.834469	0.011964	-1.812029
siz_512__143_ao	-0.013255	-9.81	9.845261	0.970822	571.823899	-0.059019	-0.694696
cpu4_6	0.013017	24.42	9.821483	0.970892	382.792403	-0.123037	-0.134469
pgalloc_dma32	-0.000023	-0.21	9.806159	0.970938	964.565292	0.037126	-1.858650
nmi_CPU0	-0.721516	-18.06	9.789204	0.970988	442.444411	0.489214	-0.311258
nr_mapped	-0.005756	-96.70	9.726872	0.971173	212.278086	0.617742	0.370878
nr_anon_pages	0.000023	730.49	**9.635033**	**0.971445**	57.601797	0.914946	0.829287
Est. kWs		776.59					

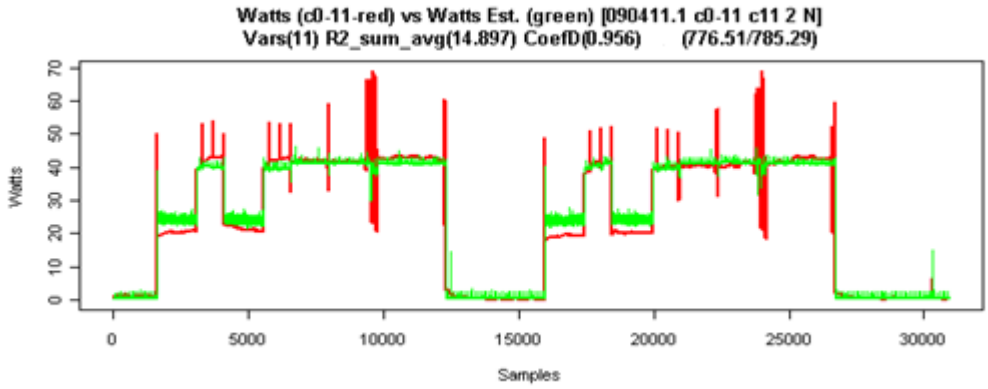
Alg. 2 Sel. Vars	x	Est. kWs	R2_sum_avg	CoefD	R2_sum_avg_x0	Corr_X0	CoefD_X0
user	0.3765905	714.24	17.517284	0.948085	17.517284	0.973702	0.948085
pgfault	0.0000764	3.13	15.439988	0.954241	957.954352	0.061616	-1.839058
pagesA	0.0000005	58.12	15.276742	0.954725	57.601510	0.914946	0.829288
pgfree	0.0000094	0.39	15.134702	0.955146	966.190532	0.024192	-1.863467
normal_3	0.0000378	5.56	15.026825	0.955465	967.667399	-0.878674	-1.867844
fil___108_no	0.0210226	2.28	14.961653	0.955659	882.308971	0.153976	-1.614870
iow	0.6478968	0.09	14.932723	0.955744	967.593784	0.006490	-1.867626
cpu3_9	0.0017003	0.58	14.906543	0.955822	948.707520	0.092729	-1.811653
cpu7_9	0.0006297	0.32	14.901006	0.955838	944.812813	0.085194	-1.800111
cpu6_9	0.0002385	0.33	14.898820	0.955845	890.794135	0.181865	-1.640017
pgalloc_normal	0.0000076	0.25	**14.896869**	**0.955851**	960.501950	0.051863	-1.846608
Est. kWs		785.29					

Alg. 3 Sel. Vars	x	Est. kWs	R2_sum_avg	CoefD	R2_sum_avg_x0	Corr_X0	CoefD_X0
user	0.3825495	725.55	17.517284	0.948085	17.517284	0.973702	0.948085
pagesA	0.0000004	46.44	17.341237	0.948606	57.601510	0.914946	0.829288
cpu5_6	0.0291761	7.53	**17.283293**	**0.948778**	499.970216	0.268409	-0.481745
Est. kWs		779.52					

Algorithm 1



Algorithm 2



Algorithm 3

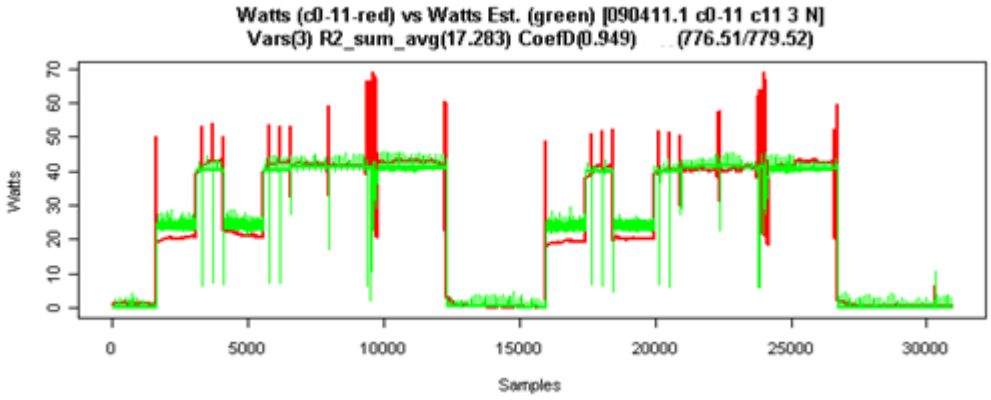


Figure 39 – Measured (red) versus estimated power dissipation (based on **Table 23**).

In **Table 23**, note that a decrease in the number of variables chosen (across algorithms) corresponds to: 1) an increase in the final R2_sum_avg and 2) a decrease in the final value of CoefD. Thus, the accuracy (in terms of R2_sum_avg) of a set of *selected variables* decreases (that is R2_sum_avg increases) when the number of variables in the set decreases. Alternatively, the proportion of variance in the data (actual power consumption) explained by the set of *selected variables* decreases as the number of variables in the set decreases. This means *Algorithm 1* consistently explains a greater proportion in the variability in the data compared to *Algorithms 2* and *3*. (Note that *Algorithm 1* also consistently selects more variables compared to the other two algorithms). **Table 24** (reprinted below) shows the accuracy of the three algorithms in estimating total energy based on *selected variables* (errors enclosed in parenthesis indicate that the model is over estimating total energy. While all three algorithms provide estimates that fall within 1.13% of measured values, *Algorithm 1* performs the best coming within 0.01%.

Table 24 – Evaluation of power estimates ($n1c1p1aX$ - Contiguous HPCC).

	Tot. kWs	Est. kWs	% Error
Algorithm 1	776.51	776.59	(0.01)
Algorithm 2	776.51	785.29	(1.13)
Algorithm 3	776.51	779.52	(0.39)

The above analysis indicates that *Algorithm 1* does a good job selecting variables for the model. However, it is also producing negative coefficients, which implies that corresponding variables contribute negative energy to the total. For instance, in **Table 23** /proc/stat.*sys* contributes -23 kWs to total energy for *Algorithm 1*. To compensate for negative Watts, other variables over contribute, such as /proc/stat.*user*, whose estimated energy contribution exceeds actual energy consumption (total energy measured for this trial is approximately 776 kWs yet /proc/stat.*user* is contributing approximately 782 kWs). An explanation for negative coefficients showing up in a model is that different variables may be double counting system activity, and

thus, the intersection of overlap has to be subtracted out. **Section 6.13** provides some evidence that supports this observation.

The primary problem introduced by negative coefficients is that it is not possible to determine or estimate each variable's individual contribution to total energy. To address this problem, *Algorithm 2* adds the following constraint to *Algorithm 1*: the first variable selected has a positive coefficient and as the algorithm proceeds, variables are selected such that the coefficients at step *i* are positive. Note in **Table 23** that all coefficients for the second section (*Algorithm 2*) are positive. This allows estimating each variable's contribution to total or actual power. This also seems to imply that the selected variables are now mutually exclusive or measuring power consumption of a component that does not overlap with another variable's measurements. Verification of this observation is left for future research (see **Section 6.13**).

Another observation in **Table 23** is that some variables selected by *Algorithms 1* and *2* are negatively correlated with measured power consumption. Recall the column labeled Corr_x0 shows how strongly that variable's utilization samples are correlated with power samples. To test whether better variables can be identified, *Algorithm 3* adds the following constraint to *Algorithm 2*: at each step, the correlation between each selected variable and power consumption has to be greater than 0.25 (using Pearson's Correlation Coefficient). [10] This constraint not only selects strongly correlated variables, it also further reduces the number of automatically selected variables. Throughout the remainder of this Chapter, *Algorithm 3* is utilized to help identify a general set of variables applicable to the entire *virgo2* cluster.

To conclude this section, note that *Algorithms 2* and *3* may omit variables such as /proc/stat.*sys* (which is selected by *Algorithm 1*) that are not strongly correlated with power consumption for an entire trial, but are important contributors to total power for at least one phase of a computation. This observation is addressed in greater detail in **Section 6.5**. Finally,

[10] Note that *Algorithm 3* reduces to *Algorithm 2* when the correlation constraint = 0. This tells the algorithm to find any correlations between -1 and 1. A correlation specified that is less than 0 tells the algorithm to search for correlations such that CORR < -x and CORR > x. A correlation cutoff greater than zero specifies that the algorithm should search for variables whose correlation is greater than the cutoff, that is CORR > x.

there is some consistency among selected variables across the three algorithms. In each case, /proc/stat.*user* and /proc/meminfo.*pagesA* (/proc/vmstat.*nr_anon_pages*) are selected. However, each algorithm also selects its own set of unique variables which requires making a decision regarding which variables to include for developing a full cluster power model. At this point, the variables highlighted in green identify an initial list of variables considered for full cluster models.

6.4 Effect of Watt window size on variable selection

Recall, each experiment in this research consists of a set of node level Watt meter samples and OS utilization counters collected from the /proc file system during the execution of the HPCC benchmarks on a subset of cores available on the *virgo2* cluster. A set of *training variables* can be automatically selected based on a full trial (as shown in the previous section) or a subset of samples contained in the trial. For instance, compare **Figure 39** with **Figure 42** and **Figure 43** (reprinted below). The **Figure 39** corresponds to a full trial with the latter two corresponding to subsets of this trial. This section explores how the selection of a Watt window size affects the variables selected, where a Watt window corresponds to a contiguous subset of samples in a trial.

Figures 42 and **43** show measured and estimated power loads after applying *Algorithm 2* to the Watt window that includes the *n*1*p*1*c*1*a*1 HPCC benchmark computation (the MPI affinity part of the experiment shown in **Figure 37**). In this case, *Algorithm 2* selects ten variables. When idle samples are eliminated from consideration as shown in **Figure 43**, the algorithm selects all but two of those selected by the previous sub-trial. (In each figure, the number of variables is given in parenthesis after "Vars" in the second line of the heading.) **Tables 25** and **26** show the least squares results that correspond to **Figures 42** and **43** respectively. The two variables selected for the first case are /proc/stat.*procs* and /proc/buddyinfo.*normal_3*.

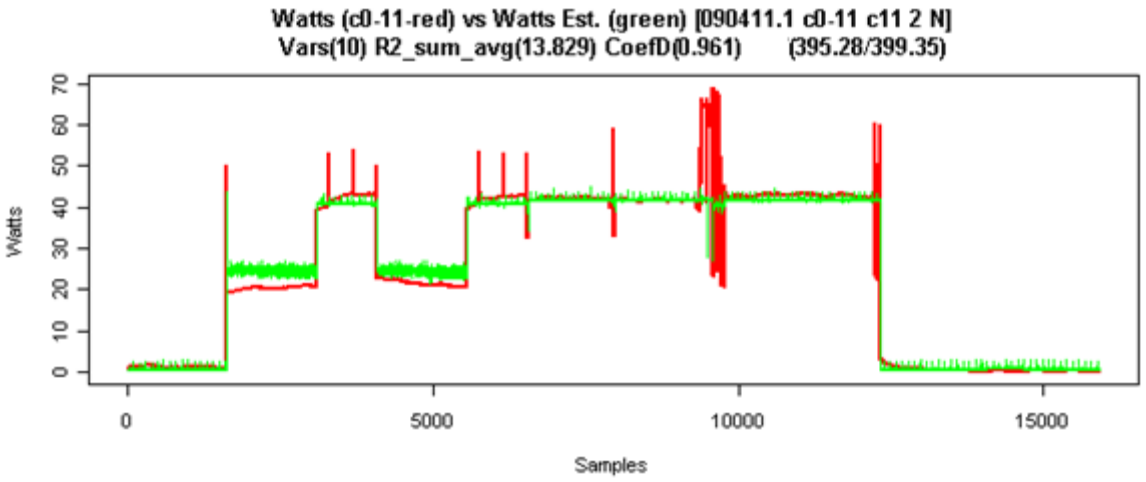


Figure 42 – Measured (red) versus estimated power load for $n1p1c1a1$.

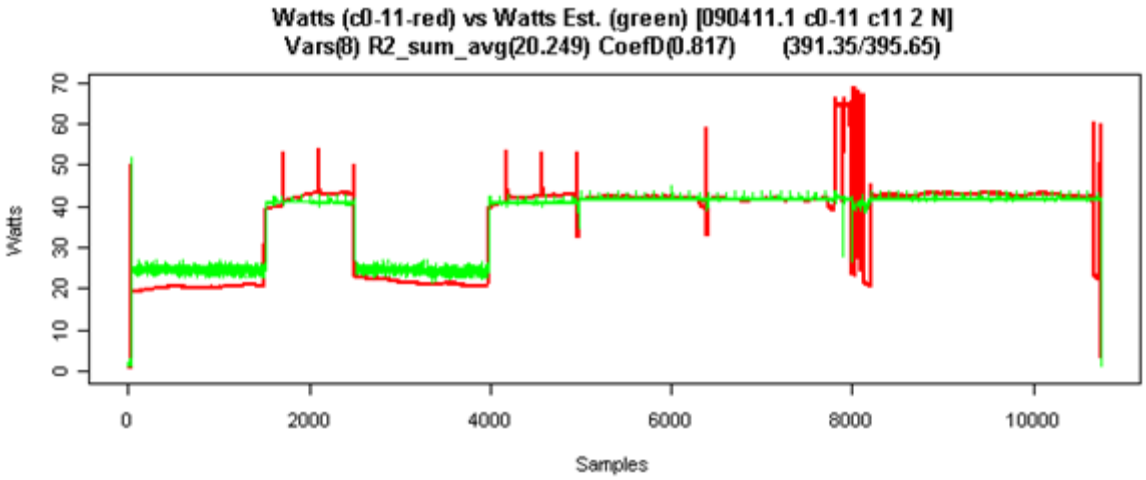


Figure 43 – Measured (red) versus estimated power load for $n1p1c1a1$ (no idle time).

Excluding samples that correspond to idle computational phases when applying *Algorithm 2* causes R2_sum_avg to increase from 13.829 to 20.249 and CoefD to decrease from 0.961 to 0.817. The reason for this is because a large number of small errors are being removed from the calculation. The error between measured power and estimated power is very small for periods where a compute node is idling. That is, when a compute node is idling, the magnitude of

the variance of both power consumption and sytem activity is low, resulting in small residual errors during these computational phases. Thus, idle time has the effect of reducing the value for R2_sum_avg when the number of idle samples is high and this corresponds to an increase in the value of CoefD. The higher value of CoefD in turn seems to indicate that the model provides a better explanation of the variance. This also implies a much better selection of variables by the algorithm. However, this is not necessarily the case because the small errors introduced by idle samples are skewing the results. This result is important to keep in mind when using R2_sum_avg and CoefD to evaluate system power models, particularly for cases where idle time is a significant portion of a computation.

6.5 Correlation between a variable and power consumption

Notice that many of the variables selected by *Algorithms 1* and *2* in **Tables 23**, **25,** and **26** have Corr_x0 values that are less than 0.25 (recall 0.25 is the cutoff or threshold for *Algorithm 3*), indicating weak correlation between the system activity measured by the variable and power consumption. The selection of these variables may occur for two reasons: 1) they reflect power consumption for a particular phase of the computation; and 2) the variables have better averaging effects that result in greater reductions in R2_sum_avg compared to other variables even though they are not strongly linked to power consumption. Clearly, the latter types of variables need to be identified and eliminated from consideration.

This section identifies three types of variables selected by *Algorithms 1* and *2*:

1. *Type 1*: Variables that are strongly correlated with power consumption and have a clear association with power consumption throughout all (most) phases of an HPCC computation.
2. *Type 2*: Variables that are weakly correlated with power consumption yet have a clear association with power consumption for a particular phase of an HPCC computation.

3. *Type 3*: Variables that are weakly correlated with power consumption and have no clear association with power consumption for any phase of an HPCC computation.

To see this, **Figure 44** through **Figure 54** show the results of applying several of the variables selected by *Algorithms 1* and *2* (shown in **Table 23**) as single variable models for power consumption. To facilitate comparisons, **Figures 44**, **45**, **50** and **51** are reprinted below. The variable's correlation with power load, Corr_x0, for a full trial is found in **Table 23** (with the exception of /proc/stat.*sys*). The variable's association with power consumption is indicated by the corresponding figure.

Notice that the *Type 2* variable /proc/stat.*sys* is not selected by *Algorithms 2* and *3*, while other *Type 2* variables are selected such as /proc/slabinfo.*fil___108_no* (perhaps because *sys* is negatively correlated with the overall power meter sample and *fil___108_no* is not). At this stage, it seems that *sys* should be thrown out of consideration in developing cluster power models, but as shown in **Section 6.9**, this is not necessarily a good idea. This is because some variables that display *Type 1* behavior for the current configuration/trial (such as *user*) show *Type 2* behavior in other trials. On the other hand, some variables categorized as *Type 2* or *Type 3* here (such as *sys* and /proc/slabinfo.*siz_102__141_ao* respectively), display *Type 1* behavior in other HPCC computations that involve more cores. Thus, more analysis is needed before selecting or discarding variables.

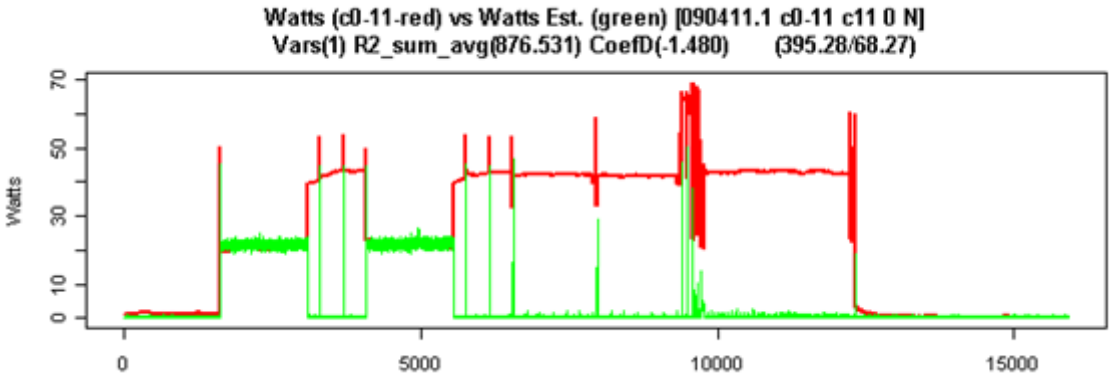


Figure 44 – /proc/stat.*sys*, *Type 2* variable.

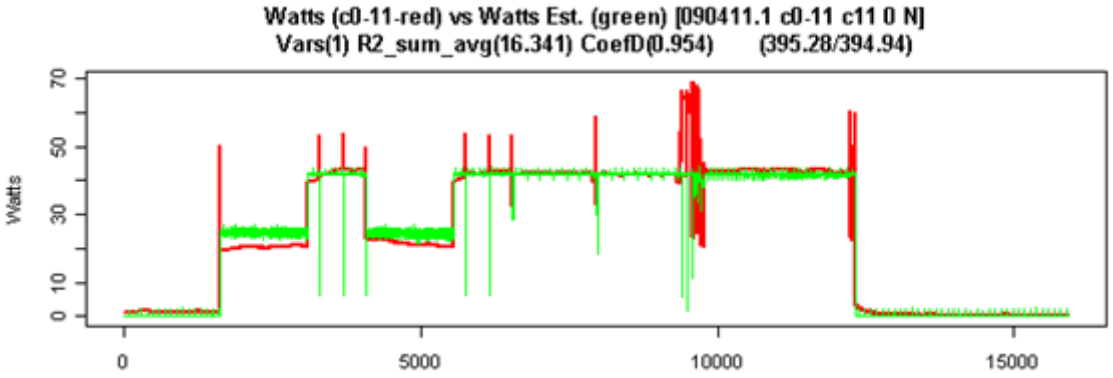


Figure 45 – /proc/stat.*user*, *Type 1* variable.

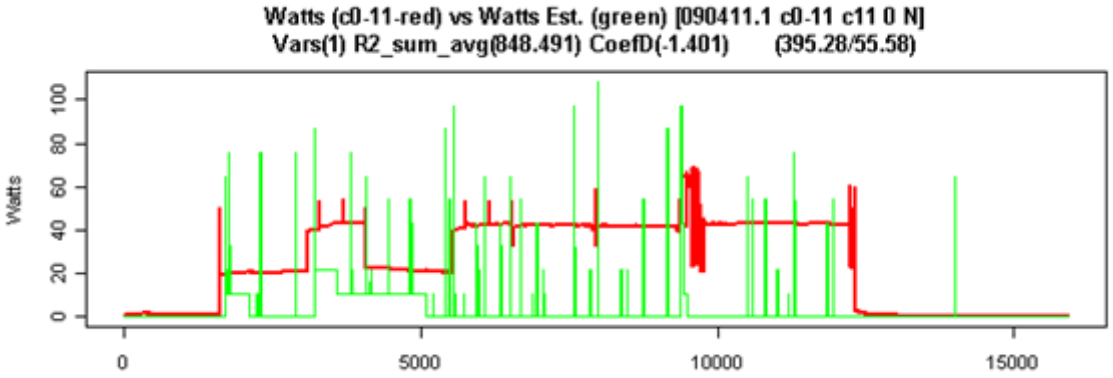


Figure 50 – /proc/slabinfo.*fil___108_no*, *Type 2* variable.

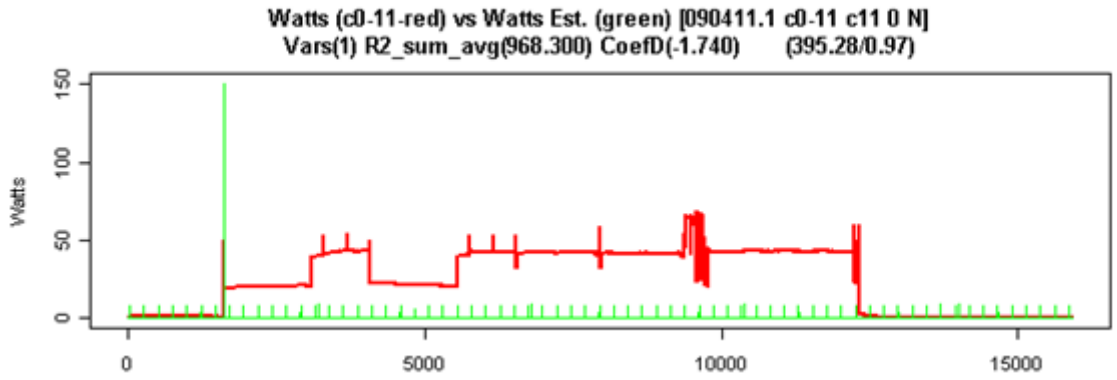


Figure 51 – /proc/stat.*procs*, *Type 3* variable.

6.6 Coefficient generation using split HPCC benchmarks

Next consider what happens when the Watt window is focused in on the final phase of the $n1p1c1a1$ computation shown in **Figure 43**. This is illustrated in **Figure 55**, where the Watt window shows only the HPL phase of the computation (each phase of the HPCC benchmarks are identified below). **Table 27** shows the output that results when *Algorithm 2* is applied to the new Watt window. There are a few things to note in this case:

- The value for R2_sum_avg is very low (less than 0.06).
- The coefficient of determination is low (0.36) indicating that this set of variables is not providing a good account for the variance in the data.
- The total energy estimate is almost equal to the total measured energy (round off shows an exact match).
- Many of the selected variables do not appear to be the most appropriate for modeling HPL power consumption. For instance, the selection of /proc/stat.*sysi* says that the time spent by the system in the idle state for the HPL phase contributed 8.8 Watts to total energy consumption. However, HPL is considered compute intensive and the system is typically not expected to be idle during this phase, especially when executed on a single core.

- The variable *commitA* contributes over 80% of the total energy consumed by HPL. This is a very unlikely scenario considering that HPL is a CPU intensive task. On the other hand, if this 80% contribution is correct, this implies that *commitA* not only measures memory utilization, it also measures processor activity, another very unlikely scenario. Recall that *commitA* is an estimate of the amount of memory that will be needed to fulfill the current needs of a running process (See **Section 3.2.5**).

One possible explanation for the collapse of *Algorithm 2* may be that there is not significant variance in the computation to accurately identify the most relevant variables. Expanded Watt windows, as shown in **Figures 42** and **43**, support this observation, where the corresponding tables (**Tables 25** and **26**) show better selections of variables. With this idea in mind, this section conducts a similar analysis as **Section 6.3**. However, in this case, split HPCC benchmark computations (recall **Section 6.3** generated training coefficients using contiguous HPCC benchmarks) are utilized to determine if increased variation in a data samples results in better variable selections overall. To accomplish this, the main program HPCC program, `hpcc.c` was modified by inserting code that causes the main process sleep for 120 seconds after the completion of each sub-benchmark. Not only does this help introduce (artificial) variation into the data, it also makes it easier to isolate each computational phase which in turn enables a more detailed analysis of each sub-benchmark's impact on power consumption (left for future work).

Figure 56 (reprinted below) shows power meter samples obtained by splitting the HPCC benchmarks and executing them on the same configuration tested in **Section 6.3**: $n1p1c1aX$. Specifically, the trial shown in the figure shows a set of power phases that correspond to an $n1p1c1a1$ computation, an idle period, and a second set of power phases that correspond to an

*n*1*p*1*c*1*a*0 computation. **Figure 57** [11] (reprinted below) identifies each of the computational phases using the *n*1*p*1*c*1*a*1 sub-trial as an example.

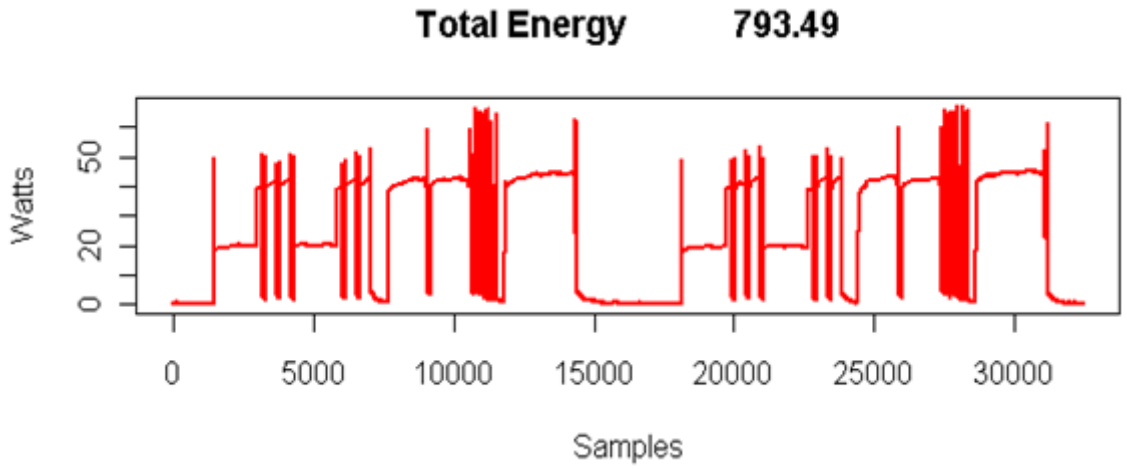


Figure 56 – Watt meter power load samples (*n*1*p*1*c*1*aX*, Split HPCC).

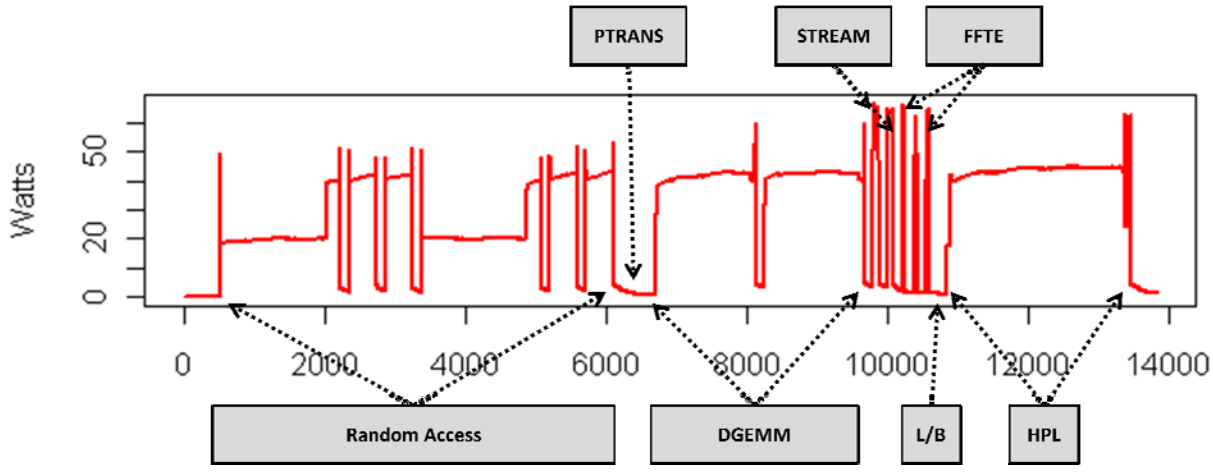


Figure 57 – Watt meter power load samples (*n*1*p*1*c*1*a*1, Split HPCC, left part of **Figure 56**).

[11] In this dissertation, PTRANS was not analyzed, which explains why its activity is idle during its computational phase. On the other hand, L/B is idle in this case because an *n*1*p*1*c*1 configuration involves a single core, which has no partner to ping pong with.

Appendix FT6 provides results obtained after applying *Algorithm*'s *1* through *3* to the split HPCC benchmarks computations shown in **Figure 56**. **Table 28** gives results in a format similar to **Table 23**, **Figure 59** illustrates split HPCC power estimates in a similar format to **Figure 39**, and **Table 29** evaluates the selected variables' performance in a similar format as **Table 24**. Note the following observations when comparing **Table 28** with **Table 23**:

- Each approach selects a largely different set of variables for each algorithm.
- A small number of variables are common between the two approaches (the intersection). For the three algorithms, contiguous and split analyses each select:

  Algorithm 1: /proc/stat.*user*, /proc/stat.*sys*, /proc/meminfo.*commitA*, /proc/meminfo.*pagesA* (equivalent to /proc/vmstat.*nr_anon_pages*) /proc/vmstat.*pgfree*, /proc/vmstat.*pgalloc_dma32*, and /proc/vmstat.*nr_mapped*

  Algorithm 2: /proc/stat.*user*, /proc/vmstat.*pgfree*, /proc/vmstat.*pgalloc_normal* and /proc/slabinfo.*fil___108_no*

  Algorithm 3: /proc/stat.*user*

The point to make is each approach is selecting similar mixes of variables of interest. Whereas **Table 23** shows the selection of *pagesA*, **Table 29** shows the selection of *pageTables*, both of which have a strong correlation to power consumption.

- Both approaches also select a mix of /proc/schedstat, /proc/interrupts, and /proc/buddyinfo variables (e.g. *cpuX_9*, *normal_X*, *nmi_CPUX*, etc.). Here, the important point to note is that each is selecting a mix of /proc/schedstat.*cpuX_9* variables. Recall, these variables track the amount of time processes spend running on processor *X* (see **Section 3.2.7**).
- Comparing the (few) unique variables selected by each approach (contiguous versus split), i.e. those selected by the contiguous HPCC trial and not selected by the split HPCC trial vice versa, the unique selections of one approach do not appear better or

worse than the other's. As a matter of fact, these selections appear to be "undesirable" and need to be further analyzed to determine whether or not they do or do not reflect power consumption accurately.

A comparison of **Table 29** with **Table 24** shows that the split and contiguous HPCC benchmark approaches tend to perform equally well in terms of estimating total power. That is, each approach selects variables whose power estimates fall within 1.25% of actual power consumption. All things being equal, the remainder of this work is based on the split HPCC computations because this approach facilitates deeper analysis of individual HPCC phases.

6.7 Adjustment of correlation threshold for Algorithm 3

The data shown in **Table 30** are the result of decreasing the correlation threshold from its default value of 0.25 to 0.001 and re-running *Algorithm 3* for the split HPCC benchmarks. In this case, the algorithm selects all variables whose correlation with power consumption is strictly greater or equal to 0.001. For the most part, **Table 30** merges the results of *Algorithm 2* and *Algorithm 3* of **Table 28**. In addition, reducing the correlation threshold improves the accuracy of the energy estimate compared to *Algorithm 2* of **Table 28**, where the estimate is 795.02 kWs compared to actual value 793.49 kWs (within 0.18%). It is left to future research to explore the details associated with selecting positively correlated variables to for power consumption models.

6.8 Factor analysis of the split HPCC benchmarks for *n*1*p*1*c*1*aX*

Up to this point, this analysis has focused on applying *Algorithms 1*, *2* and *3* to system activity measured by *target variables* for *n*1*p*1*c*1*aX* computations of the HPCC benchmarks. As shown in the previous sections, however, these algorithms tend to select varying subsets of *target* variables. One explanation for this may be that the algorithms are selecting different but "related

variables" that measure utilization of the same system component but perhaps in a different way. Consider for example /proc/stat.*user* and /proc/schedstat.*cpuX_9*. Whereas *user* measures time spent executing in user mode and is the sum across each processor, *cpuX_9* variables measure total time spent by all tasks executing on CPU *X* only (see **Sections 3.2.7** and **3.2.9**).

To identify variables that may overlap or measure utilization of the same system components, this section applies factor analysis to all *target variables* related to the split *n*1*p*1*c*1*aX* HPCC trial. The goal is to narrow down the mix of variables for defining the cluster model. It is important to note that factor analysis is utilized simply as a tool for narrowing down the list of variables and for identifying those that show patterns that are strongly related to power consumption. No attempt is made to confirm factors identified in this work.

As discussed in **Section 5.6.2**, factor analysis can be used to find patterns in data (Rummel, 2012). This section uses factor analysis to identify /proc variables that are related to common "unobserved" factors. For instance, /proc/meminfo.*pagesA* provides an (observed) measurement of anonymous page utilization. One *unobserved* factor may be the power consumption of underlying physical memory modules when anonymous pages are accessed. A second *unobserved* factor may be the power the CPU consumes when anonymous pages are accessed. Factor analysis is applied to *target* variables for each /proc file in the current work and each analysis also includes Watt meter samples. Including Watt meter samples in factor analysis of *target* variables aims to find groups of variables that not only describe a component's utilization but can also be linked to that component's power consumption. In the current work, subsets of variables that are collectively linked to a factor through what are called factor loadings (explained below) are referred to as a *cluster* of variables related to the factor.

Tables 31 through **40** show output obtained after applying factor analysis to the variables listed in the first column of each table. With the exception of **Table 36**, each table lists variables that are tracked by the corresponding /proc file. Included in each table are at least one column of factor loadings. Factor loadings are the correlation coefficients between the variables and the factors (Garson, 2012). They measure how much a variable is related to a factor (Rummel,

2012). To illustrate, consider **Table 35** (reprinted below for easy reference). Here we see five columns of factor loadings (with columns labeled Factor1 through Factor5) for /proc/meminfo variables. Note the following:

- Using a loading cutoff of 0.6, two clusters of variables are identified (which are highlighted so they can be easily distinguished). The first cluster, in the "Factor1" column, includes variables *active*, *pagesA*, *pageTables*, *commitA*, and *mapped***.** The loadings indicate that these variables are strongly correlated with the factor. The second cluster, "Factor2" includes *cache*, *slab* and *buf*. The remaining columns do not indicate other clusters of variables for /proc/meminfo.

Table 35 – Factor loadings for /proc/meminfo ($n1p1c1aX$, Split HPCC).

Variable	Factor1	Factor2	Factor3	Factor4	Factor5
active	**0.99**	0.05	0.10	0.08	-0.06
pagesA	**0.99**	0.04	0.13	0.08	-0.05
pageTables	**0.98**	0.04	0.12	0.10	-0.05
commitA	**0.98**	0.04	0.13	0.10	-0.04
Watts	**0.91**	0.06	0.07	0.13	-0.02
mapped	**0.63**	0.16	-0.05	**0.75**	-0.03
cache	-0.01	**0.96**	0.17	-0.09	0.17
slab	0.10	**0.90**	0.26	0.05	0.11
buf	-0.08	**0.69**	-0.53	-0.22	-0.40
inactive	0.14	**0.27**	**0.88**	0.12	0.32
dirty	0.01	0.05	0.00	0.02	-0.01
memF	**-0.98**	-0.06	-0.13	-0.08	0.04
nfs	-0.01	0.00	0.00	0.00	0.01
writeb	0.00	0.00	0.02	0.00	0.00

- "Factor1" includes *Watts* (shown bold), whose factor loading is 0.91. This shows that the cluster of variables in this column includes Watt meter samples. The second cluster in the Factor2 column does not include Watt samples, where Watts loading is

0.06. This indicates that the unobserved factor measured by *slab*, *buf* and *cache* is not contributing significantly to power consumption.

- Note *memF* has a factor loading of -0.98. This indicates that this variable characterizes the opposite of what the cluster is associated with. This makes sense because free memory is simply total available memory minus all allocated memory.

The fact that the factor loading for *Watts* is high for Factor1 indicates that *active*, *pagesA*, *pageTables*, *commitA* and to a lesser extend *mapped* (its factor loading is 0.63) are measuring the utilization of an "unobserved" factor that is strongly associated with power consumption. Three of these variables will be considered in cluster power models: *pagesA*, *pageTables*, and *commitA*. (The reason for excluding *active* is explained in **Section 6.10**). Factor analysis results for all other /proc files show that some sets of /proc files identify clusters that include power consumption and other files identify other clusters that do not include power consumption (more on this below).

At this point it is relevant to ask whether the clusters of variables shown in **Tables 31** through **40** are actually related. A simple way to verify the results of factor analysis is to apply each variable in a cluster as a single variable model of power consumption and to compare results graphically. **Figure 61** through **Figure 66** show single variable results (based on the *n*1*p*1*c*1*aX* split HPCC trial) for some of the variables included in **Table 35** (/proc/meminfo), and **Figure 67** through **Figure 72** show single variable results for some of the variables included in **Table 39** (/proc/stat). Note the following:

- *mapped* (**Figure 65**) least resembles the other variables in its cluster (it also has the lowest factor loading of 0.63 shown in **Table 35**).
- *memF* (**Figure 66**) is clearly not related to variables grouped into the cluster for Factor1, where its factor loading is -0.98.
- *user* (**Figure 67**) is the only variable in /proc/stat that clusters with *Watts*.

Table 41 (reprinted below) compiles results from **Figure 61** through **Figure 72**. The table includes R2_sum_avg, CoefD and "% Err", which is the percent difference between measured active energy for the trial 793.49 (labeled "Tot. kWs" and shown in the last row of the table), and estimated active energy, "Est. kWs". Note that the variables with the smallest errors meet the following for **Table 31** through **Table 40**: the variables have high factor loadings (above 0.9) when *Watts* also has a high factor loading.

Of the variables shown in **Table 41** *user* is the one that most closely reflects power consumption, with an R2_sum_avg value of 20.27, CoefD value of 0.94 and energy estimate the falls within 0.7% of actual energy. Note the inclusion of *sys*. On the surface, this variable appears to have very little relation to power consumption. However as indicated previously (also reference **Figure 44** and **Figure 68**), it is a *Type 2* variable that is clearly associated with power consumption for at least one HPCC phase. The importance of /proc/stat.*sys* as both a metric that describes system utilization and as a predictor of power consumption is made evident in the next section.

Table 41 – Factor analysis of meminfo and stat (*n*1*p*1*c*1*aX*, Split HPCC).

Variable	R2_sum_avg	CoefD	Est. kWs	% Err
active	52.19	0.85	773.50	2.52
pagesA	54.43	0.84	759.06	4.34
pageTables	51.89	0.85	771.07	2.83
commitA	55.55	0.84	761.02	4.09
(mapped)	198.37	0.43	831.18	4.75
memF	889.47	-1.56	187.18	76.41
user	20.27	0.94	787.94	0.70
sys	864.00	-1.48	131.67	83.41
sysi	805.57	-1.32	324.61	59.09
Tot. kWs			**793.49**	

Analysis of the remaining variables from **Table 31** through **Table 40** reveals that the variables most closely related to power consumption (that is, those with factor loading greater

than 0.9 when Watts also has a factor loading greater than 0.9) are those shown in bold in **Table 41** (*active*, *pagesA*, *pageTables*, *commitA*, and *user*). Note that **Table 40** (which shows variables tracked by /proc/vmstat) includes *nr_anon_pages* and *nr_page_table_pages*. These two correspond to *pagesA* and *pageTables* (see **Sections 3.2.5** and **3.2.10**). Finally, recall these results are based on the *n*1*p*1*c*1*aX* trial.

Factor analysis is not just useful for identifying variables that are strongly related to power consumption. This tool also reveals other variables that tend to form groups or clusters related to other "unobserved" factors. For example, as shown previously, **Table 35** shows that *slab*, *cache* and *buf* form a cluster under Factor2. These types of clusters help to identify variables related to other system components (that perhaps consume less power during an HPCC computation) as shown later in this section and in **Sections 6.9** through **6.11**.

This section concludes by summarizing some of the major clusters of variables identified for the split *n*1*p*1*c*1*aX* computations. Factor analysis was applied to the entire set of *target* variables to identify those that form clusters and thus potentially track utilization of the same "unobserved" system components or processes. To illustrate the procedure, consider **Table 36** (reprinted below) which shows factor analysis results for variables in /proc/stat, /proc/interrupts, and /proc/net/dev. [12] As it relates to the primary network interface, Ethernet 0 or *eth0*, for *virgo2* compute nodes, two types of clusters are identified. The first cluster, under Factor1 of the table, describes Ethernet related interrupt activity, where factor loadings for /proc/stat.*int1*, /proc/stat.*int100*, and /proc/interrupts.*eth0_CPU6* are at least 0.94. The second cluster, under Factor2, includes transmitted (/proc/net/dev.*eth0_Tx_Pkt*) and received (/proc/net/dev.*eth0_Rx_Pkt*) packets, with factor loadings 0.99 and 0.98 respectively.

[12] Factor analysis in R requires at least three variables. In order analyze /proc/net/dev to show that transmitted and received packets form a cluster, it was necessary to include a few other variables. Based on prior analysis of variables in /proc/stat and /proc/interrupts, the variables included in a factor analysis of /proc/net/dev were chosen because of their relation to received and transmitted packets.

Table 36 – Factor loadings of net/dev and stat (*n*1*p*1*c*1*aX*, Split HPCC).

Variable	Factor1	Factor2	Factor3
int100	**0.98**	0.12	0.11
eth0_CPU6	**0.98**	0.12	0.11
int1	**0.94**	0.12	0.11
eth0_Tx_Pkt	0.10	**0.99**	0.04
eth0_Rx_Pkt	0.12	**0.98**	-0.11
lo_Tx_Pkt	0.09	0.00	0.03
Watts	0.00	0.00	-0.01

As it relates to /proc/interrupts, the variables *eth0_CPUX* (where *X* is a core id from 0 through 7) track interrupts generated by *eth0* that are serviced by that core (**Section 3.2.3**). For data collected in this research, the vast majority of interrupts are serviced by CPU6 (processor or slot 0, core 6) and CPU7 (processor or slot 1, core 7). As indicated previously, factor analysis of **Table 36** variables shows a link between /proc/stat.*eth0_CPUX* and the two /proc/stat variables, *int1* and *int100*. That is, these variables form a cluster that is related to activity generated by *eth0*. This observation is applied later in **Section 6.10**, where /proc/stat.*int1* or /proc/stat.int*100* is the variable selected to represent network activity for full cluster models.

A similar process as described above was applied to the identification of other clusters of variables that may be linked to other system components such as the processor, memory or disk. Because the process involves generation of many tables as well as ad hoc analysis, only the results of the analysis are provided. **Table 42** provides a set of named sections that include a bulleted list of variables (the clustered variables) related to that section through factor analysis. A variable is included in the category if its factor loading is greater than 0.6. Note that the results apply to the *n*1*p*1*c*1*aX* computations. **Table 53**, described in the next section, shows results obtained for *n*20*p*40*c*160*aX* computations.

6.9 Factor analysis of the split HPCC benchmarks for *n*20*p*40*c*160*aX*

This section applies factor analysis to data collected from the two metered compute nodes (c0-11 and c0-12) for an *n*20*p*40*c*160*aX* trial. Results are provided in **Tables 43** through **51**. In these tables only the first two factors are included per node (columns 4, 5 for c0-11 and columns 6, 7 for c0-12). Also included in the tables are the first two factors for *n*1*p*1*c*1*aX* results obtained in the previous section (columns 2, 3 for c0-11). Each pair of columns is identified using the notation "*c*1*w*11" (short for configuration *n*1*p*1*c*1*aX* applied to c0-11 Watt meter data). Clusters are defined as variables in a column with a factor loading of at least 0.7 for this analysis (note: the cell's background for these variables is highlighted to clearly show the clusters). There are a few points to mention regarding these tables:

- Some clusters increase in size as the number of cores increases from 1 to 160. Consider **Table 44**. Note the increase in cluster size as the number of cores increases from 1 to 160. This suggests an increase in component activity related to the number of cores which is captured by the variable(s).
- Some clusters highly associated with power consumption in the *c*1 case, become less associated for *c*160, whereas others not highly associated with power consumption increase as the number of cores increase. For instance, **Table 50** shows that /proc/stat.*user* is the only variable that belongs to the cluster of variables that includes *Watts* for the 1 core case (recall *user* is a *Type 1* variable for *n*1*p*1*c*1*aX*). When the number of cores increases to 160 (for the *c*160*w*11 columns), the cluster associated with *Watts* changes significantly, dropping *user* and adding six other variables (*sys*, *hirq*, *sirq*, *int1*, *int100*, and *procs_r*). This indicates that some variables change roles as it relates to the three types discussed in **Section 6.5** depending on the number of cores the HPCC benchmarks are run on. These variables are readily identified by comparing factor loadings as the number of cores increase in **Table 43** through **Table 51** for cases where *Watt* loadings are relatively high.

- Next */proc/meminfo* (**Table 47**) and */proc/vmstat* (**Table 51**) clusters include the same variables but their association with power consumption drops slightly: *Watts* loading factor for /proc/meminfo drops from 0.91 for a 1 core computation to 0.88 for 160 cores.
- Some variables that are not associated with power consumption for a 1 core computation remain unassociated regardless of the increase in the number of cores involved in a computation. This is a strong indicator of their relatively low impact on power consumption and by extension, their exclusion from full cluster power modeling.

Figure 70 through **Figure 80** show single variable models of c0-11 data to illustrate cluster memberships. Note that **Table 48** suggests that *eth0_CPU7* and the two /proc/net/dev variables (*eth0_Rx_Pkt* and *eth0_Tx_Pkt*) are not part of the same cluster. However, **Figures 72**, **73**, and **74** suggest at least a small association. Close inspection of **Table 48** reveals an increase in the relationship between the variables, where factor loadings under Factor1 for *eth0_Rx_Pkt* and *eth0_Tx_Pkt* increase from 0.12 to 0.29 and 0.10 to 0.29 respectively (see *c*1*w*11 Factor1 loadings). This establishes a relationship between /proc/stat.*int1* and /proc/stat.*int100* and network IO which is exploited in the next section to specify a full cluster model.

A similar factor analysis was conducted as in Section 6.8 to identify clusters of variables. Results are provided in **Table 53**. In this case, factor analysis of the 160 core trial (*n*20*p*40*c*160*aX*) does not provide results as clearly as the 1 core trial (*n*1*p*1*c*1*aX*). For example, there are some instances where /proc/stat.*sys* and /proc/stat.*user* are both strongly associated with power consumption and other instances where only /proc/stat.*sys* is strongly associated. This is particularly true when /proc/stat.*user* and /proc/stat.*sys* are included in factor analyses that also include /proc/meminfo variables.

6.10 Algorithm 2 results across all *nXaX* experiments

This section describes the selection of six variables that are applied to full cluster power modeling in the current work. The section begins by presenting results obtained from application of *Algorithm 2* to data obtained for each metered compute node for each configuration in **Table 21**. Recall some configurations were tested multiple times and most trials include power measurements (for $n > 1$) from compute nodes c0-11 and c0-12. These cases are referenced by appending "*tKwC*" to the configuration string, where *K* denotes the *K*th trial for the configuration and *C* identifies the compute node whose power was measured.

6.10.1 Selected variables across experiments

Tables 54 through **67** show the results of applying *Algorithm 2* to each configuration. **Table 67** is reprinted below and shows results for trial *n*20*p*40*c*160*aX*-*t*2*w*11. In addition to information reported in previous tables (coefficients, R2_sum_avg, CoefD, etc.) each table includes the first two factors obtained when factor analysis is applied to the variables in the tables. A factor loading of 0.8 is used as the cutoff for clusters of variables that include Watts. These are highlighted throughout the tables. Finally, **Table 68** (reprinted below) shows how well selected variables model actual power loads and total energy as it relates to **Tables 54** through **67**. That is, if we select each table as a single node power model (of metered node *wX*) based on the configuration, **Table 68** shows how well the variables in each table are trained. In most cases, energy estimates are within 1.6% of measured energy for that configuration, with the best set of training variables appearing in **Table 67** where the energy estimate falls within 0.02%.

Close inspection of **Tables 54** through **67** reveals that no single variable is always selected across all configurations. These tables also illustrate the difficulty of identifying a general set of variables that can be applied across configurations to produce a consistent system level model. There is too much inconsistency from one configuration to the next in the variables selected and it is not clear which table shows the best set of variables that can be applied to all

configurations. Recall, *Algorithm 2* is selecting the set of variables based on data for that trial that results in a (near) minimal R2_sum_avg. Thus, applying the variables from **Table 67** to configuration *n*1*p*1*c*1*aX*-*t*1*w*11 will not necessarily result in a lower error than that presented in **Table 54**.

Close inspection of **Tables 54** through **67**, on the other hand, does reveal some variables are selected more frequently across configurations than others. **Figure 81** shows how many times each variable is selected across experiments with **Table 69** showing which variables are selected once or twice. The five most frequently appearing variables are /proc/stat.*user* (11), /proc/interrupts.*ide0_CPU1* (8), /proc/stat.*sys* (7), /proc/vmstat.*nr_dirty* (7), and /proc/meminfo.*active* (7). The following variables appear at least six times: *nr_anon_pages*, *cpu5_4*, *cpu6_9*, *bio___86_ao*, *commitA*, *usb_4_CPU4*. Note the following as it relates to the frequencies of variables selected:

- *nr_anon_pages* is identical to /proc/meminfo.*pagesA* (which appears 4 times). This means anonymous pages actually appears 10 times across configurations. Recall, program data and stack space are stored in anonymous pages (Kay, 2011).

Table 67 – Least Squares results for *Algorithm 2* (*n*20*p*40*c*160*aX*-*t*1, c0-11).

Alg. 2 Sel. Vars	x	kWs	R2_sum_avg	CoefD	R2_sum_avg_x0	Corr_X0	CoefD_X0		Factor1	Factor2
cpu6_9	0.000008	0.08	129.0229	0.9651	129.0229	**0.9829**	**0.9651**		**0.92**	0.34
active	0.000000	1.18	74.5631	0.9799	848.3530	**0.8791**	**0.7708**		0.57	0.61
user	0.126594	173.39	69.9082	0.9811	5008.3452	0.4840	-0.3534		0.08	**0.99**
siz_204__139_ao	0.126113	236.96	64.0782	0.9827	504.4810	0.9300	0.8637		**0.77**	0.41
eth0_CPU7	0.000919	118.61	60.1614	0.9837	2320.6158	**0.7165**	**0.3729**		**0.88**	-0.12
sys	0.095826	626.85	58.3022	0.9842	1102.0219	**0.8577**	**0.7022**		**0.97**	-0.07
nr_anon_pages	0.000025	140.75	56.9641	0.9846	942.8141	**0.8857**	**0.7452**		**0.58**	0.61
pgmajfault	0.815577	0.12	56.6789	0.9847	7544.8925	0.0026	-1.0388		0.00	0.00
nfs	0.138905	0.07	56.6502	0.9847	7541.1402	0.0151	-1.0378		0.00	0.01
ide0_CPU1	0.014399	1.06	56.6425	0.9847	4947.3052	0.4323	-0.3369		0.35	0.19
usb_4_CPU1	0.013602	0.22	56.6394	0.9847	7117.1466	0.1705	-0.9232		0.15	0.04
lo_Rx_Pkt	0.025895	0.44	56.6371	0.9847	7057.0785	-0.0001	-0.9070		0.00	0.00
dirty	0.000119	0.73	56.6363	0.9847	3190.2701	0.6263	0.1379		0.31	0.53
dma32_0	0.000001	0.43	56.6363	0.9847	5158.5922	-0.4753	-0.3940		-0.03	-0.68
Watts		1300.88							**0.86**	**0.43**

Table 68 – *Algorithm 2* percent error in energy estimates across all trials.

Configuration / Node / Experiment	Tot. kWs	Est. kWs	% Error
(*n1p1c1aX-t1w11*)	793.49	803.44	1.25
(*n1p1c4aX-t1w11*)	863.25	877.35	1.63
(*n1p2c8aX-t1w11*)	356.32	358.00	0.47
(*n1p2c8aX-t2w11*)	350.62	356.14	1.57
(*n1p2c8aX-t3w11*)	342.84	346.91	1.19
(*n2p4c16aX-t1w11*)	551.46	558.75	1.32
n2p4c16aX-t1w12	554.98	551.28	0.67
(*n4p8c32aX-t1w11*)	908.72	912.61	0.43
(*n4p8c32-t1w12*)	794.41	798.43	0.51
(*n4p8c32-t2w11*)	893.25	900.70	0.83
(*n4p8c32-t2w12*)	794.27	801.74	0.94
(*n20p30c160aX-t1w11*)	1295.75	1302.34	0.51
n20p40c160-t1w12	1222.86	1221.51	0.11
n20p40c160aX-t2w11	1301.19	1300.88	0.02

Table 70 – System Components and variables strongly correlated with power consumption.

System Component	Variables
processor/CPU	*user* *sys* *cpuX_9*
memory	*active*, *commitA*, *pagesA* (*nr_anon_pages*), *pageTables* (*nr_page_table_pages*), *siz_204__139_ao*, *skb_fcl_c_90_ao*, *skb_hea_c_91_ao*
network	*int1* and *eth0_CPU7*

- *usb_4_CPU4* clusters with *int76* as shown in the *Disk Utilization* sections of **Tables 42** and **53**. Together with /proc/diskstats variables, these appear a total of 16 times. While *int76* is not one of the *selected variables* in any of the tables (**Tables 54** through **67**), the fact that it belongs to a cluster related to disk utilization means it can be used to represent disk activity in cluster power models.
- *cpuX_4* variables appear a total of 15 times for various values of *X*, where *X* denotes a cpu core numbered 0 through 7. Recall, /proc/schedstat.*cpuX_4* variables count the number of times the active queue has at least one other process on it.

- *cpuX_9* variables appear a total of 31 times for various values of *X*, where *X* denotes a cpu core numbered 0 through 7. Recall, /proc/schedstat.*cpuX_9* variables measure sum of all time tasks spend running.
- *normal_X* variables appear a total of 15 times and *normal_X* tends to cluster with /proc/stat.*sysi*, time the processor spends in the idle state (reference the *Time processor spent idling* sections of **Table 42** and **Table 53**).
- Ethernet related variables appear the fewest number of times at 8: /proc/interrupts.*eth0_CPU6* (2), /proc/interrupts.*eth0_CPU7* (3), and /proc/stat.*int1* (1), /proc/net/dev/*eth0_Rx_B* (2). While *int100* is not one of the selected variables in any of the tables and *int1* is selected only once, they can be used to represent network related activity based on factor analysis results shown in **Sections 6.8** and **6.9**.

6.10.2 Selected variables strongly related to power consumption

For each experiment, there is a set of variables that more closely mirror power consumption compared to the remaining variables. These correspond to clusters of variables that include Watts when a factor loading of at least 0.8 is used as a cutoff in **Tables 54** through **67**. These variables include: *active*, *commitA*, *pagesA* (*nr_anon_pages*), *pageTables* (*nr_page_table_pages*), *cpuX_4* where $X \in \{1, 3, 4, 5, 7\}$, *cpuX_9* where $X \in \{0, 1, \ldots, 7\}$, *user*, *sys*, *int1*, *procs_r*, *siz_204__139_ao*, *skb_fcl_c_90_ao*, and *skb_hea_c_91_ao*. Of these, *user* typically contributes significantly to power consumption for configurations that involve a fewer number of cores and *sys* contributes significantly to power consumption for larger numbers of cores. Some variables tracked by /proc/slabinfo begin to strongly correlate with power consumption for larger numbers of cores. **Table 70** (reprinted above) lists these variables and the system component to which they are related.

6.10.3 General system level power consumption model

Based on analysis conducted in this chapter, the variables chosen to model power consumption of a compute node are shown in **Table 71**. As it relates to processor utilization, /proc/stat.*user* and /proc/stat.*sys* are two variables that have been shown to be strongly linked to power consumption. These variables are also useful for measuring the power efficiency. That is, the ratio of *user* to *sys* provides a measure of the degree to which power is put to use calculating HPCC results versus put to use managing system resources.

Table 71 – System Components and variables final list.

System Component	Variables
processor/CPU	/proc/stat.*user* /proc/stat.*sys*
memory	/proc/meminfo.*commitA* /proc/meminfo.*pagesA* /proc/meminfo.*pageTables*
disk	/proc/stat.*int76*
network	/proc/stat.*int100* or /proc/stat.*int1*

The three variables that measure memory utilization are selected because these show consistent correlation with power consumption. While /proc/meminfo.*active* also displays consistent power utilization, **Figure 78** illustrates why this variable is eliminated from consideration. Note how the minimum value for power consumption continually increases as time samples progress, something *commitA*, *pageTables*, and *pagesA* do not do (see **Figures 75**, **76**, and **77**). This behavior may introduce undesirable results later. Finally, while the three /proc/slabinfo variables in **Table 70** are strongly associated with power consumption for the 160 core case (see **Figure 79** and **Figure 80**), they are not included because they are not consistently correlated with power consumption compared to the former variables. Finally, the disk utilization and network utilization variables are selected because they are tracked in a single /proc file (/proc/stat) and they have been shown to have an association with the respective device. Note

that the final selections in **Table 71** are found in just two /proc files, which simplifies data collection for modeling full cluster power consumption.

6.11 System level power models for metered compute nodes

This section defines two cluster power models based on the variables given in **Table 71**. These are labeled *Model A* and *Model B*. The variables included in these models are:

Model A: *user*, *sys*, *int76*, *int100*, *commitA*

Model B: *user*, *sys*, *int76*, *int100*, *pagesA*, *pageTables*

The system activity associated with the variables above is used to *train* each model or alternatively derive the coefficients that define the model. In the following analysis, the coefficients are based on power measurements and system activity for the two metered compute nodes. The result is two sets of coefficients per model, one set for c0-11 and another for c0-12, denoted $\boldsymbol{x}_{c0\text{-}11,A}$ and $\boldsymbol{x}_{c0\text{-}12,A}$ for *Model A* and $\boldsymbol{x}_{c0\text{-}11,B}$ and $\boldsymbol{x}_{c0\text{-}12,B}$ for *Model B*.

To estimate system level energy for each model, coefficients are first calculated using **Eq. 5.1** (reprinted below for easy reference). The values of the coefficients are listed in **Table 76**. Second, an estimate of power consumption, $\boldsymbol{b}'_M$, is calculated using **Eq. 5.2** (reprinted below). Note that $\boldsymbol{b}_C$ is replaced by $\boldsymbol{b}'_M$ to show the power estimate is for the metered node *M*. Third, the energy estimate in kWs, E'_M, is obtained by summing all power values of $\boldsymbol{b}'_M$ as shown in **Eq. 6.1** (reprinted below):

Eq. 5.1 $$\boldsymbol{x}_M = (\boldsymbol{A}_M^{\mathrm{T}}\boldsymbol{A}_M)^{-1}\,(\boldsymbol{A}_M^{\mathrm{T}})\,\boldsymbol{b}_M$$

Eq. 5.2 $$\boldsymbol{b}'_M = \boldsymbol{A}_M \bullet \boldsymbol{x}_M$$

Eq. 6.1 $$E'_M = \sum_{k=1}^{N} b'_{M,k}$$

Where

- $\boldsymbol{x}_M$ = set of model coefficients for a metered compute node.
- $\boldsymbol{A}_M$ = a metered compute node's activity matrix formulated using the model variables.
- $\boldsymbol{b}_M$ = power load samples for a metered compute node
- $\boldsymbol{b}'_M$ = estimated power load samples for a metered compute node
- $\boldsymbol{b}'_{M,k}$ = the kth power estimate of compute node M.
- N = number of data points (samples in the analysis).

Table 72 – *Model A* and *B* training results across all trials.

		Training Model A				Training Model B		
config:node		Tot. kWs	Est. kWs	% Error		Tot. kWs	Est. kWs	% Error
*n*1*p*1*c*1-*t*1*w*11		793.49	778.78	1.85		793.49	790.91	0.33
*n*1*p*1*c*4-*t*1*w*11		863.25	824.18	4.53		863.25	863.13	0.01
*n*1*p*1*c*8-*t*1*w*11		350.10	334.22	4.54		350.10	348.94	0.33
*n*1*p*1*c*8-*t*2*w*11		350.62	333.61	4.85		350.62	347.42	0.91
*n*1*p*1*c*8-*t*3*w*11		342.84	331.51	3.30		342.84	(343.68)	0.25
*n*2*p*4*c*16-*t*1*w*11		551.46	535.72	2.85		551.46	547.89	0.65
*n*2*p*4*c*16-*t*1*w*12		544.98	519.87	4.61		544.98	538.24	1.24
*n*4*p*8*c*32-*t*1*w*11		908.72	873.46	3.88		908.72	905.10	0.40
*n*4*p*8*c*32-*t*1*w*12		794.41	792.11	0.29		794.41	(796.22)	0.23
*n*4*p*8*c*32-*t*2*w*11		893.25	866.10	3.04		893.25	892.11	0.13
*n*4*p*8*c*32-*t*2*w*12		794.27	792.07	0.28		794.27	(796.00)	0.22
*n*20*p*40*c*160-*t*1*w*11		1295.75	1274.43	1.65		1295.75	1290.07	0.44
*n*20*p*40*c*160-*t*1*w*12		1222.86	1198.99	1.95		1222.86	1215.48	0.60
*n*20*p*40*c*160-*t*2*w*11		1301.19	1277.98	1.78		1301.19	1294.88	0.48
*n*20*p*40*c*160-*t*2*w*12		1222.86	1198.99	1.95		1222.86	1215.48	0.60
Absolute Average				2.76				0.45

A measure of how well the model variables train the model is obtained by calculating the percent difference between measured energy and estimated energy as shown in **Eq. 6.2** in **Section 6.2**. **Table 72** (reprinted above) shows results. Included in the table are measured energy "Tot. kWs", estimated energy "Est. kWs", and the percent error in the total estimate (% Error) for each configuration. In all cases, a full trial includes MPI with and without affinity, so the notation *aX* is not included in the configuration strings for simplicity. The table shows *Model A* is less accurate than *Model B* across all trials. That is, the coefficients of *Model A* approximate total energy within 4.85% in all cases whereas *Model B* estimates are within 1.24%. Overall, both models perform well, with absolute average errors of 2.76% and 0.45% respectively for *Model A* and *Model B* (shown in the last row of the table).

The coefficients can now be used to predict (model) power loads and energy consumption for an arbitrary compute node's system activity. To evaluate how well coefficients generated on one compute node estimate the power load of a different compute node (as will be done in the next section to estimate *virgo2* energy), the following procedure is applied. For trials in **Table 72** whose configuration has a value of n = 2, 4 or 20, coefficients calculated on c0-11 are applied to system activity for c0-12 which results in a modeled power load for c0-12. Next, the energy for the modeled power load is compared to the actual energy measured for the measured power load on c0-12. Similarly, coefficients generated on c0-12 are applied to c0-11 system activity and results are also compared similarly and shown in **Table 73** (reprinted below). Note that the table actually evaluates three sets of coefficients for each model:

- Coefficients trained using c0-11 data: $x_{\text{c0-11,A}}$ and $x_{\text{c0-11,B}}$
- Coefficients trained using c0-12 data: $x_{\text{c0-12,A}}$ and $x_{\text{c0-12,B}}$
- The average of the coefficients for each model: $x_{\text{avg,A}}$ and $x_{\text{avg,B}}$

For example, from *Model A*, two sets of coefficients are generated, $\boldsymbol{x}_{c0\text{-}11,A}$ and $\boldsymbol{x}_{c0\text{-}12,A}$. A third set of coefficients is obtained by taking the average of the two: $\boldsymbol{x}_{avg,A} = (\boldsymbol{x}_{c0\text{-}11,A} + \boldsymbol{x}_{c0\text{-}12,A})$ /

2. The three coefficients for *Model A* are then applied to system activity for compute nodes c0-11 and c0-12. This results in energy estimates that are compared to actual energy from a node's measured power consumption. In **Table 73**, cells that are not highlighted show percent error in training estimates. These are the same values contained in **Table 72** for configurations with $n > 1$. Modeled estimates are shown in cells that are highlighted.

Table 73 is organized into four sections. The top two sections shows results when *Model A* coefficients are applied to c0-11 and c0-12 activity with the bottom two sections providing results for *Model B*. Each average shown at the bottom of each section is computed by taking the average of each row that corresponds to the label. For instance, in the last column of the section labeled "*Model A* Coefficients applied to c0-12", the value 2.93 for $\boldsymbol{x_{\mathrm{Avg}}Avg}$ is the average of 3.20, 4.74, 4.28, 1.14, and 1.28, which correspond to the errors obtained when applying the averaged coefficients, $\boldsymbol{x_{Avg,A}}$, to c0-12 system activity. Note the following regarding the table:

- For *Model A*, the maximum error is 12.70% with the minimum 1.14%.
- For *Model B*, the maximum error is 14.80% with the minimum 0.22%.
- Energy estimates of c0-11 are less accurate than corresponding estimates of c0-12 energy. In other words, coefficients trained with compute node c0-11 data do better at estimating c0-12 energy.
- On average, *Model A* predicts power within 7.39% whereas *Model B* predicts power within 7.63%, not considering the averaged coefficients. (7.39% is the average of diagonals 8.95% and 5.83% found in the bottom rows of the top section with a similar argument for 7.63%)
- In most cases, the averaged coefficients provide the best energy estimates. The averaged errors for *Model A* are 4.36% (average of 5.80 and 2.93) whereas those for Model B are 3.86%.

Table 73 – Evaluation of energy estimates (*nXaX*, *Model A* and *Model B*).

	***Model A* Coefficients applied to c0-11**			***Model A* Coefficients applied to c0-12**		
config:coeffs	**Tot. kWs**	**Est. kWs**	**% Error**	**Tot. kWs**	**Est. kWs**	**% Error**
*n*2*c*16-*t*1:11	551.46	535.72	2.85	**544.98**	**535.24**	**1.79**
*n*2*c*16-*t*1:12	**551.46**	**520.30**	**5.65**	544.98	519.87	4.61
*n*2*c*16-*t*1:Avg	**551.46**	**528.01**	**4.25**	**544.98**	**527.56**	**3.20**
*n*4*c*32-*t*1:11	908.72	873.46	3.88	**794.41**	**(872.07)**	**9.78**
*n*4*c*32-*t*1:12	**908.72**	**793.35**	**12.70**	794.41	792.11	0.29
*n*4*c*32-*t*1:Avg	**908.72**	**833.40**	**8.29**	**794.41**	**(832.09)**	**4.74**
*n*4*c*32-*t*2:11	893.25	866.10	3.04	**794.27**	**(864.46)**	**8.84**
*n*4*c*32-*t*2:12	**893.25**	**793.51**	**11.17**	794.27	792.07	0.28
*n*4*c*32-*t*2:Avg	**893.25**	**829.81**	**7.10**	**794.27**	**(828.27)**	**4.28**
*n*20*c*160-*t*1:11	1295.75	1274.43	1.65	**1222.86**	**(1274.48)**	**4.22**
*n*20*c*160-*t*1:12	**1295.75**	**1199.49**	**7.43**	1222.86	1198.99	1.95
*n*20*c*160-*t*1:Avg	**1295.75**	**1236.96**	**4.54**	**1222.86**	**(1236.74)**	**1.14**
*n*20*c*160-*t*2:11	1301.19	1277.98	1.78	**1222.86**	**(1277.94)**	**4.50**
*n*20*c*160-*t*2:12	**1301.19**	**1199.53**	**7.81**	1222.86	1198.99	1.95
*n*20*c*160-*t*2:Avg	**1301.19**	**1238.76**	**4.80**	**1222.86**	**(1238.47)**	**1.28**
		$x_{c0\text{-}11}$ ***Avg***	2.64		$x_{c0\text{-}11}$ ***Avg***	**5.83**
		$x_{c0\text{-}12}$ ***Avg***	**8.95**		$x_{c0\text{-}12}$ ***Avg***	1.82
		x_{Avg} ***Avg***	**5.80**		x_{Avg} ***Avg***	**2.93**

	***Model B* Coefficients applied to c0-11**			***Model B* Coefficients applied to c0-12**		
config:coeffs	**Tot. kWs**	**Est. kWs**	**% Error**	**Tot. kWs**	**Est. kWs**	**% Error**
*n*2*c*16-*t*1:11	551.46	547.89	0.65	**544.98**	**(549.35)**	**0.80**
*n*2*c*16-*t*1:12	**551.46**	**536.42**	**2.73**	544.98	538.24	1.24
*n*2*c*16-*t*1:Avg	**551.46**	**542.16**	**1.69**	**544.98**	**543.80**	**0.22**
*n*4*c*32-*t*1:11	908.72	905.10	0.40	**794.41**	**(907.08)**	**14.18**
*n*4*c*32-*t*1:12	**908.72**	**795.65**	**12.44**	794.41	(796.22)	0.23
*n*4*c*32-*t*1:Avg	**908.72**	**850.37**	**6.42**	**794.41**	**(851.65)**	**7.21**
*n*4*c*32-*t*2:11	893.25	892.11	0.13	**794.27**	**(891.64)**	**12.26**
*n*4*c*32-*t*2:12	**893.25**	**796.06**	**10.88**	794.27	(796.00)	0.22
*n*4*c*32-*t*2:Avg	**893.25**	**844.08**	**5.50**	**794.27**	**(843.82)**	**6.24**
*n*20*c*160-*t*1:11	1295.75	1290.07	0.44	**1222.86**	**(1285.68)**	**5.14**
*n*20*c*160-*t*1:12	**1295.75**	**1218.24**	**5.98**	1222.86	1215.48	0.60
*n*20*c*160-*t*1:Avg	**1295.75**	**1254.16**	**3.21**	**1222.86**	**(1250.58)**	**2.27**
*n*20*c*160-*t*2:11	1301.19	1294.88	0.48	**1222.86**	**(1290.41)**	**5.52**
*n*20*c*160-*t*2:12	**1301.19**	**1218.46**	**6.36**	1222.86	1215.48	0.60
*n*20*c*160-*t*2:Avg	**1301.19**	**1256.67**	**3.42**	**1222.86**	**(1252.94)**	**2.46**
		$x_{c0\text{-}11}$ ***Avg***	0.42		$x_{c0\text{-}11}$ ***Avg***	**7.58**
		$x_{c0\text{-}12}$ ***Avg***	**7.68**		$x_{c0\text{-}12}$ ***Avg***	0.58
		x_{Avg} ***Avg***	**4.05**		x_{Avg} ***Avg***	**3.68**

The results shown in **Table 73** indicate that overall, *Model B* coefficients perform well at predicting energy with the best estimates produced using the average of the coefficients. For this case, the maximum energy estimation error is 7.21% with the minimum 0.22%.

A third model, named *Model C*, was considered to determine if an improvement would be obtained over *Model B* by eliminating *pageTables*. *Model C* was compared to *Models A* and *B* for the last trial shown in **Table 72** for both metered compute nodes. The model variables are

Model C: *user*, *sys*, *int76*, *int100*, *pagesA*

Results are shown in **Table 74** (reprinted below) which includes *Models A* and *B* for easy reference. Here we see *Model C* actually performs worse than both *Models A* and *B*. This indicates that *commitA* is a better predictor of power consumption than *pagesA*. On the other hand, combining *pagesA* with *pageTables* improves results over just using *commitA*. **Table 75** (reprinted below) shows results when coefficients are applied to *Model C* in a similar manner as **Table 73** (also shown are *Model A* and *B* results for easy reference). Once again, in all cases, *Model B* performs the best. Thus, a full cluster power model can be based on *Model B* using the averaged coefficients resulting from training models to compute c0-11 and c0-12 data. In general, this approach can be used to predict power consumption for a full cluster by measuring power consumption on two or more compute nodes, fitting *Model B* variables to power consumption and system activity on those nodes, then taking the average of the resulting coefficients. The averaged coefficients in turn are used to model power loads for the entire cluster by applying the coefficients system activity matrices comprised of data from two /proc files: /proc/stat and /proc/meminfo.

The next section applies coefficients derived for *Models A*, *B* and *C* to estimate power consumption per variable as well as power consumption across the entire cluster.

Table 74 – Training coefficient energy estimates (*n*20*c*160*aX*-*t*2, *Models A*, *B*, and *C*).

Model A			
Node	Tot. kWs	Est. kWs	% Err.
c0-11	1301.19	1277.98	1.78
c0-12	1222.86	1198.99	1.95

Model B			
Node	Tot. kWs	Est. kWs	% Err.
c0-11	1301.19	1294.88	0.48
c0-12	1222.86	1215.48	0.60

Model C			
Node	Tot. kWs	Est. kWs	% Err.
c0-11	1301.19	1274.78	2.03
c0-12	1222.86	1197.25	2.09

Table 75 – Fitted variable energy estimates (*n*20*c*160*aX*-*t*2, *Models A*, *B*, and *C*).

	Model A			Model B			Model C		
config:coeffs	Tot. kWs	Est. kWs	% Err.	Tot. kWs	Est. kWs	% Err.	Tot. kWs	Est. kWs	% Err.
c0-11 data									
n20c160e2:11	1301.19	1277.98	1.78	1301.19	1294.88	0.48	1301.2	1274.8	2.03
n20c160e2:12	1301.19	1199.53	7.81	1301.19	1218.46	6.36	1301.2	1200.2	7.76
n20c160e2:Avg	1301.19	1238.76	4.80	1301.19	1256.67	3.42	1301.2	1237.5	4.89
c0-12 data									
n20c160e2:11	1222.86	1277.94	-4.50	1222.86	1290.41	-5.52	1222.9	1270.4	-3.89
n20c160e2:12	1222.86	1198.99	1.95	1222.86	1215.48	0.60	1222.9	1197.3	2.09
n20c160e2:Avg	1222.86	1238.47	-1.28	1222.86	1252.94	-2.46	1222.9	1233.8	-0.90

6.12 Full Cluster Power model

This section compares full cluster energy estimates of the three models defined in the previous section. **Table 76** shows the coefficients for *Models A*, *B*, and *C*. Each set of coefficients is applied to system activity matrices for 20 compute nodes based on the second trial of an *n*20*p*40*c*160*aX* computation (this is the same trial shown in the last two rows of **Table 81**). Note for this configuration, the number of samples for c0-11 is 20,984 while the number of samples for c0-12 is 20,436. These values are used to calculate total idle energy at the end of this section. Active energy estimates (based on active power loads) are given in **Tables 77**, **78**, and **79**. Note that each table begins with a summary of the errors for the fitted variables. Recall, the coefficients shown in **Table 76** are the result of training models with c0-11 and c0-12 data.

Before discussing results, note that a slight problem arises for the data set related to the *n*20*p*40*c*160*aX* computation. There are two sets of Watt meter samples for this computation that are offset by a significant amount (approximately 1,600 samples) as shown in **Figure 82**. This is due to downloading Watt meter data from the two Watt meters at two time points 1,600 seconds apart after the trial. In this case, the Watt window(s) of interest are identified by the black slider bars below each graph. Thus, to estimate energy for each compute node that corresponds to these windows, it is necessary to align /proc data to the windows. That is, before applying coefficients, the proper alignment needs to be found between each compute node's /proc samples and the targeted Watt meter samples. At this point, the question that arises is *which Watt window should data be aligned to*? For the purposes of this research, data are aligned relative to each Watt window (which ideally should produce very similar estimates and fortunately is the case). Thus, in **Table 76**, the first set of estimates are based on aligning data to the Watt window for c0-11 and the second set of estimates is based on aligning data to the Watt window for c0-12. (Another fortunate thing about this problem is that it only applies to this evaluation because for arbitrary system activity, power estimates can be based on any /proc utilization window or sample.)

Focusing back on the results in **Tables 77**, **78**, and **79**, the first column below the summary identifies a compute node. The top labels identify which coefficients are applied and the labels "aligned to" identify which Watt meter sample /proc data are aligned to. Below each of these labels are energy estimates and errors with respect to total measured energy in kWs ("Tot. kWs" in the table summary) for c0-11. (Note that errors can alternately be computed relative to c0-12, which should produce different values with similar patterns). The last row of each column (T/AE) is the sum of energy estimates across nodes immediately followed by the average of all errors for that column. Note the following with respect to these tables:

- Each set of coefficients produces similar energy estimates and errors across nodes. This indicates that system activity measured by the model variables is consistent across nodes. In other words, system activity during an HPCC computation does not

vary significantly across nodes for an *n*20*p*40*c*160*aX* computation which suggests that the HPCC benchmarks are producing homogenous system activity across nodes.

- *Model B* produces the smallest errors of the three models across all trials, but in this case it results in errors that are between those of the other two coefficients.

Table 80 – Totals of modeled power consumption per compute node (*Models A*, *B* and *C*).

c0-11 coefficients applied				
	c0-11 aligned		c0-12 aligned	
Model	Est. kWs	% Err	Est. kWs	% Err
A	25530.15	1.897	25529.06	1.901
B	25856.41	0.643	25851.64	0.662
C	25448.96	2.209	25448.16	2.212

c0-12 coefficients applied			
c0-11 aligned		c0-12 aligned	
Est. kWs	% Err	Est. kWs	% Err
23963.89	7.915	23963.13	2.315
24337.71	6.479	24333.36	6.496
23967.51	7.902	23966.86	7.904

Averaged coefficients applied			
c0-11 aligned		c0-12 aligned	
Est. kWs	% Err	Est. kWs	% Err
24747.04	4.906	24746.10	4.910
25097.06	3.561	25092.50	3.579
24708.27	5.055	24707.52	5.058

For this work, energy estimates for the entire cluster are based on the averaged coefficients for *Model B* (consistent with results from the previous section) applied to data aligned to c0-11. **Table 80** (reprinted above) aggregates the totals for each model from **Tables 77**, **78**, and **79**. From here, the total active energy estimate for the *virgo2* cluster is 25,097.06 kWs. Idle (or baseline) power averages 155.1 Watts on c0-11 and 157.4 Watts on c0-12 with the average 156.25. This means baseline total energy for 20 compute nodes and one frontend is approximately 21 * (20,984) * 156.25 / 1000 = 68,853.75 kWs (recall, 20,984 is the number of samples collected for the trial). Thus, total energy (the sum of total active and baseline) is 25,097.06 + 68,853.75 = 93,950.81 kWs. Converting kilowatt seconds to kilowatt hours (see **Section 2.1**) yields 26.1 kWh to run the second split *n*20*p*40*c*160*aX* trial. At 0.8 cents per kWh, the total cost of the trial is $2.09 with an annual estimated cost of $3,137.66 (assuming this type of computation characterizes yearly activity). At 0.14 cents per kWh, the total cost of the trial is $3.65 with an annual estimated cost of $5,490.91. 73.3% of total energy (and cost) is baseline energy (the minimum amount of power consumed by the system under normal operating conditions with no compute node down time) while 26.7% is the result of computation.

6.13 Motivation for future research

This section briefly explores ideas for future research. Three analyses are discussed: 1) estimate of each variable's energy contribution to total energy consumption; 2) fitting system activity of non-metered nodes to power consumption of metered nodes; and 3) analysis of negative coefficients.

6.13.1 Estimating a variable's energy contributions to total energy consumption

In this section, power contributions of each model variable (which implies the variable is measuring activity of a hardware component) were estimated for the metered compute nodes for the *n*20*p*40*c*160*aX*-*t*2 trial. The coefficients in **Table 76** were used to estimate the individual contributions to total energy. Results are given in **Table 81**, in which each variable's contribution to total energy calculated as specified using **Eq. 6.4**. The information provided in this table disaggregates energy estimates for *Models A*, *B*, and *C* derived in **Section 6.11** (recall those models were trained using data from metered compute nodes). The coefficients from **Table 76** were then applied to system activity for all remaining compute nodes (all non-metered nodes). **Table 82** shows the energy for each variable, $E_{C,i}$ averaged across all nodes, *C*, when the model coefficients, $\boldsymbol{x}_M$, are used to estimate power loads per variable, $b_{C,i}$, based on system activity on those nodes (see related discussion in **Section 6.2**). **Table 81** and **Table 82** are organized similarly. Model variables are given in the first column. Next are the model coefficients, $\boldsymbol{x}$, estimated energy "Est. kWs", and each variable's percent contribution to the total estimate, "Est. kWs%". Each table is split into three sections, one section for each model. The row labeled "Node_Est" under each section gives system level energy estimates based on each of the three coefficients and the row labeled "Cluster_Est" provides results for the full cluster obtained by multiplying "Node_Est" by 20, thus these tables are extrapolating system level results to the full cluster. Note the following as it relates to *Model B*:

- *user* and *sys* are estimated to contribute between 73% and 80% of total power. This indicates that these two variables are measuring more than processor activity.
- *int76* (disk activity) is estimated to contribute less than 0.05% of total power on average.
- *int100* (network activity) is estimated to contribute between 6% and 7% of total energy.
- *pagesA* and *pageTables* contribute between 13% and 20% of total energy.

Results indicate that each variable may be tracking system utilization linked to power consumption of more than one component. For instance, it is very likely that *user* and *sys* reflect power consumption of the processor, memory, and motherboard. Future research can be aimed at determining the mix of hardware components each variable measures that can be linked to power consumption.

6.13.2 Fitting system activity to power consumption of metered nodes

Coefficients for all non-metered compute nodes were fitted to c0-11 and c0-12 power measurements. In this case, 18 sets of coefficients were calculated for each model. This set of coefficients was then applied to estimating each compute node's estimated power consumption as well as each variable's individual contributions. **Table 83** and **Table 84** show power estimates for each compute node with **Table 85** and **Table 86** giving results relative to each coefficient. The latter two tables include the original values of the coefficients, the averaged coefficients across all nodes, the standard deviations of the coefficients, power contribution and percentage contribution. Note the following as it relates to **Table 83** and **Table 84** :

- Once again, *Model B* provides the lowest errors per node.
- Average total active energy for *Model B* is 25,120.95 kWs compared to 25,097.06 kWs obtained in the previous section, values that are very close.

Note the following as it relates to **Table 85** and **Table 86**:

- The coefficient values for *user* for *Models A* and *C* are around 0.14, whereas all other *user* values are near 0.12.
- /proc/meminfo variables show the smallest standard deviation across all models, where all variables show a standard deviation of at most 0.000077.
- The standard deviation is less than 0.025 for all variables.
- The standard deviation of *int76* is always the highest, which is in the neighborhood of 0.02.

Analysis of the coefficients generated by fitting system activity of non-metered compute nodes to power consumption may be useful in identifying more accurate values for the coefficients. If coefficients can be identified more accurately, better estimates can be obtained. This also is left for future work.

6.13.3 Negative coefficients

The last evaluation provided addresses negative coefficients. Note what happens when the three /proc/meminfo variables (*commitA*, *pagesA*, *pageTables*) are combined with the /proc/stat variables (*user*, *sys*, *int76*, *int100*) and coefficients are calculated for c0-11. This is referred to as *Model D* and results are shown in **Table 87** (reprinted below). Here we see *commitA* has a negative coefficient which results in a very large negative energy contribution (-4,951 kWs). Recall, *commitA* (short for Commited_AS as presented in /proc/meminfo) is an estimate on the amount of RAM that is needed to ensure an out of memory (OOM) does not occur. Results in the table seem to identify overlap between *commitA*, *pagesA* and *pageTables*. That is, *commitA*'s coefficient appears to be subtracting out the overlap, and thus one possible explanation for negative coefficients. Future work can further address this observation. In a

similar vein, future work can explore the degree to which variables are measuring mutually exclusive system activity.

Table 87 – Coefficients and estimated energy for *Model D*.

Model D	x	Est. kWs
user	0.018532	27.57
sys	0.007757	52.43
int76	0.253472	6.16
int100	-0.000368	-48.21
pagesA	0.000077	1894.41
pageTables	0.047709	4057.36
commitA	-0.000194	-4951.25

6.14 Chapter summary

This chapter identifies six variables (*Model B*) that can be applied to predicting power consumption for HPC clusters. Results are based what is called utilization-based power models using least squares minimization problems. In the process of deriving a common set of variables, one obstacle encountered is that different coefficients that describe power consumption emerge across analyses depending on a variety of factors including the number cores involved in an HPCC computation and the number of samples analyzed.

To handle the vast amounts of data collected for the current research, a variety of tools are applied to first eliminate irrelevant variables from consideration (as indicated in **Chapter 5**) then to organize remaining variables into meaningful set that can be applied to a full cluster. The tools used in this chapter include Pearson's correlation coefficient, the average of the sum of squared residuals, the coefficient of determination and factor analysis.

Results indicate that *Model B* provides a good starting point for estimating full cluster power consumption. Four *Model B* variables are tracked by /proc/stat (*user*, *sys*, *int76* and *int100*) with the remaining two, *pagesA* and *pageTables*, found in /proc/meminfo. In most cases,

active energy estimates fall between 2 and 9 percent of "actual" measured energy with better results observed when using the average of two sets of training coefficients. *Model B* variables form the basis for full cluster energy estimates, which show the *virgo2* cluster consumes approximately 7.87 kWh computing the split HPCC benchmarks with a specified problem size N = 90,000.

Motivation for future work is also discussed which includes three analyses: 1) estimate of each variable's power contribution to total energy; 2) fitting variables to system activity of non-metered nodes and power consumption of metered nodes; and 3) analysis of negative coefficients.

Chapter 7: Conclusion

7.1 Summary

In this work full cluster utilization-based power model and analysis is conducted. The work makes use of two *watts up? Pro* power meters connected in series to two compute nodes on the *virgo2* power limited HPC cluster. Because each meter and each compute node are independent of each other, power meter samples and utilization counters are tracked using asynchronous clocks. Power meter data is downloaded from a Watt meter for each experiment conducted. OS level utilization data is collected from ten kernel specific files in the /proc file system that provide data to user land processes in a variety of formats. Data from these files is (consistently) sampled, parsed, linearized, and logged for various hardware configurations running the HPCC challenge benchmarks. Nearly 3,000 variables, constants, and configurable values are sampled per second per compute node from the ten /proc files selected for analysis. Hardware configurations involve 1, 4, 8, 16, 32 or 160 cores which correspond to 1, 2, 4 or 20 compute nodes (each compute node with 2 processors and 4 cores per processor).

Once data is collected from Watt meters and each compute node participating in an HPCC computation, the data are read into the R environment and processed. Constants are removed from /proc samples using R scripts and power meter samples and utilization counters are aligned using an algorithm implemented in R. Several R GUIs are developed to facilitate data management. In addition, three algorithms are implemented in R and are made available to the primary power modeling GUI. The algorithms automatically search a set of utilization counters and output a set of variables that provides the best "fit" for a set of power load samples resulting from an HPCC computation. These algorithms automate the trial-and-error method for analyzing and testing various combinations of variables.

In combination with mathematical tools that include 1) correlation, 2) the average of the sum of the squared residuals, 3) the Coefficient of determination, and 4) Factor analysis, the R GUI is used to analyze and select a common set of variables that can be applied to modeling

power consumption for the entire *virgo2* cluster and for other clusters in general. All models described in the current work are characterized by the set of variables that make up an *activity matrix* (which is a sample of the utilization of a compute node) and the models are defined by the resulting values of the variables, the coefficients of a least squares minimization problem.

7.2 Results

Power modeling indicates that utilization counters available in the /proc file system provide an effective means of estimating the energy consumption of a full HPC compute node. Further, results show that the best set of variables selected for a model, i.e. those that produce the minimum R2_sum_avg, vary based on a number of factors that include: The size of the Watt window selected for analysis, the hardware configuration of the HPCC computation; etc.

The best full cluster power estimates are obtained for a six variable model, *Model B*, that includes four /proc/stat counters (*sys*, *user*, *int76*, and *int100*, where *sys* and *user* correspond to CPU utilization, *int76* corresponds to disk utilization, and *int100* corresponds to network interface utilization) and two /proc/meminfo variables (*pageTables* and *pagesA*). Further, for this model, the best results are obtained taking the average of two sets of coefficients that result from training two individual models on two compute nodes. The averaged coefficients result in energy estimation errors that range from 0.22% for an *n*2*p*4*c*16 HPCC computation to 7.21% for an *n4p*8*c*32 computation. For an *n*20*p*40*c*160 configuration (which corresponds to computing the HPCC benchmarks on the entire cluster), the error in estimated energy is 2.46%.

Comparing results between *Models A, B* and *C* with analyses conducted throughout the current work shows that increasing the number of variables provides more accurate models of a single compute node. However, increasing the accuracy of a model does not necessarily produce a set of variables that provides a better characterization of the compute nodes system activity related to power consumption. Thus a tradeoff exists between model accuracy and explanatory power. Also, when using an algorithm based on minimizing the squared residuals in least squares models, the variables selected are not necessarily generalizable to an entire cluster. This

necessitates the use of other methods that can be applied to finding a representative set of variables (for instance factor analysis).

7.3 Contributions

Some of the contributions of this work include:

- Development of method for sampling /proc files that:
    1) Extracts utilization information from files that present data to user land processes in various formats.
    2) Maps each /proc sample to a single record that simplifies data collection in log files.
    3) Operates efficiently from both a performance and power consumption perspective. Efficient operation is demonstrated by applying a worst case sampling of all possible values that can be meaningfully extracted from each proc file.
    4) Ensures periodic samples are taken consistently at the middle of each sampling period.
- Development of an effective automated procedure to synchronize log files generated by power meters and compute nodes. Recall, ach compute node and power meter collects samples based on an independent clock source which results in various log files that need to be synchronized prior to analysis.
- Observation that /proc file system provides utilization data that can be used to develop utilization-based power model of an entire HPC cluster.
- Identification of three types of variables based on link between system utilization and power loads:
    4. Variables that are strongly correlated with power consumption and have a clear association with power consumption throughout all (most) phases of an HPCC computation.

5. Variables that are weakly correlated with power consumption yet have a clear association with power consumption for a particular phase of an HPCC computation.
6. Variables that are weakly correlated with power consumption and have no clear association with power consumption for any phase of an HPCC computation.

- Observation that factor analysis is useful for identifying variables that are strongly related to power consumption and helps to reveal variables that can be categorized in general groupings or clusters based on some related "unobserved" factor.
- Observation that a GUI interface, such as that implemented in R in this work, both simplifies and facilitates model development and analysis.

7.4 Future work

Throughout this work, opportunities for future research have been identified. The following is a summary of planned future work:

- Develop a framework to verify power estimates obtained for a full cluster based on training models on a subset of nodes applied to utilization information across the entire cluster.
- Determine degree to which coefficients capture power load on non-metered compute nodes.
- Determine whether negative coefficients are the result of subtracting out "overlapping" utilization measurements.
- Determine degree to which variables are mutually exclusive.
- Develop a distributed run-time power load model based on run-time calibration of model coefficients.

7.4 Conclusion

In summary, this work shows that power modeling and analysis based on the /proc file system is not only feasible it is also simple to implement and provides accurate energy estimates of both compute nodes and an entire cluster. The /proc file system is a convenient interface that allows collection of general utilization information that does not require instrumenting benchmarks or software and can be applied more generally than other approaches. In addition, the R programming environment provides a powerful, cost effective set of tools that facilitates analysis and model development.

References

Power Modeling

(Bircher and John, 2007)

L. Bircher, L.K. John. Complete System Power Estimation: A Trickle-Down Approach Based on Performance Events. IEEE International Symposium on Performance Analysis of Systems and Software. Proceedings, pp. 158-168. 2007. Retrieved 03/26/2012 from

http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4211032&tag=1

(Carol, 2007)

T. Carrol. Usage-Based Model of Computer Power Consumption. Working Paper. Department of Economics, University of Washington. December 7, 2007. Retrieved 03/26/2012 from

http://www.econ.washington.edu/user/startz/482/Papers/F07%20Second%20Draft/EconometricsPaper.pdf

(Contreras and Martonosi, 2005)

G. Contreras, M. Martonosi. Power Prediction for Intel XScale Processors Using Performance Monitoring Unit Events. Proceedings of the 2005 International Symposium on Low Power Electronics and Design, pp. 221-226. ACM. 2005. Retrieved 03/26/2012 from

http://dl.acm.org/citation.cfm?id=1077657

(Fan et al., 2007)

X. Fan, W. Weber, L.A. Barroso. Power Provisioning for a Warehouse-sized Computer. Proceedings of the 34th Annual International Symposium on Computer Architecture, pp. 13-23. ACM. 2007. Retrieved 03/26/2012 from

http://dl.acm.org/citation.cfm?id=1250665

(Feng, 2003)

W. Feng. Making a Case for Efficient Super Computing. Queue, Volume 1, Issue 7, October 2003, pp 54-64. ACM. Retrieved 03/26/2012 from

http://dl.acm.org/citation.cfm?id=957772

(Feng and Cameron, 2006)

W. Feng, K. Cameron. The Green500 List. Power Measurement of High-End Clusters. Ver. 0.1, 2006. Ver. 0.1, November 2006. Green500.org. Retrieved 03/26/2012 from

http://www.green500.org/docs/pubs/RunRules_Ver0.9.pdf

(Feng et al., 2005)

X. Feng, R. Ge, and K.W. Cameron. Power and Energy Profiling of Scientific Applications on Distributed Systems (IPDPS 05). In *19th IEEE International Parallel and Distributed Processing Symposium,* Denver, CO. 2005. Retrieved 03/26/2012 from

http://dl.acm.org/citation.cfm?id=1053727.1054376

(Heath et al., 2006)

T. Heath, A.P. Centeno, P. George, L. Ramos, Y. Jaluria, R. Bianchini. Mercury and Freon: Temperature Emulation and Management for Server Systems. Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 106-116. ACM. 2006. Retrieved 03/26/2012 from

http://dl.acm.org/citation.cfm?id=1168872

(Hsu and Feng, 2005)

C. Hsu, W. Feng. A Power-Aware Run-Time System for High-Performance Computing. Conference on High Performance Networking and Computing. Proceedings of the 2005 ACM/IEEE conference on Supercomputing, page 1, 2005. IEEE Computer Society. Retrieved 03/26/2012 from

http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1559953

(Isci and Martonosi, 2003)

C. Isci, M. Martonosi. Runtime Power Monitoring in High-End Processors: Methodology and Empirical Data. Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture, p. 93. IEEE Computer Society. 2003. Retrieved 03/26/2012 from

http://dl.acm.org/citation.cfm?id=956567

(Joseph and Martonosi, 2001)

R. Joseph, M. Martonosi. Run-Time Power Estimation in High Performance Microprocessors, In Proceeding of the International Symposium on Low Power Electronic Device, 2001. Retrieved 03/26/2012 from

http://dl.acm.org/citation.cfm?id=383119

(Kamil et al., 2008)

S. Kamil, J. Shalf, E. Strohmaier. Power Efficiency in High Performance Computing. Proceedings of the 2008 Conference on High-Performance, Power Aware (HPPAC 2008), 2008. Retrieved 03/26/2012 from

http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4536223

(Kansal et al., 2010)

A. Kansal, F. Zhao, J. Liu, N. Kothari, A.A. Bhattacharya. Virtual machine power metering and provisioning. In Proc. of SOCC, 2010. Retrieved 03/26/2012 from

http://dl.acm.org/citation.cfm?id=1807136

(Lewis et al., 2008)

A. Lewis, S. Ghosh, N.F. Tzeng. Run-time Energy Comsumption Estimation Based on Workload in Server Systems. Workshop on Power Aware Computing and Systems, HotPower 2008. Proceedings, December 7, 2008. Retrieved 03/26/2012 from

http://www.usenix.org/event/hotpower08/tech/full_papers/lewis/lewis_html/

(Li and John, 2003)

T. Li, L.K. John. Run-time modeling and estimation of operating system power consumption. 2003 ACM SIGMETRICS International Conference on Measurement and Modeling of Computing Systems. ACM. 2003. Retrieved 03/26/2012 from

http://dl.acm.org/citation.cfm?id=885651.781048

(Lungu et al., 2009)

A. Lungu, P. Bose, D.J. Sorin, S. German, G. Janseen. Multicore Power Management: Ensuring Robustness via Early-Stage Formal Verification. Proceedings of the 7th IEEE/ACM International Conference on Formal Methods and Models for Codesign, pp. 78-87. IEEE Press. 2009. Retrieved 03/26/2012 from

http://portal.acm.org/citation.cfm?id=1715759.1715771&coll=GUIDE&dl=GUIDE

(Mathew, 2004)

B.K. Mathew. The Perception Processor. School of Computing, University of Utah. Ph. D. Dissertation. May, 2004. Retrieved 03/26/2012 from

http://www.siliconintelligence.com/people/binu/perception/

(Mandagere and Du, 2009)

N. Mandagere, D. Du. Data Center Power Management. Digital Technology Center Intelligent Storage Consortium (DISC). ACM Transactions on Computational Logic, Vol. V, No. N, September 2009. Retrieved 03/26/2012 from

http://www-users.cselabs.umn.edu/classes/Fall-2011/csci8980-ass/files/power-management/SurveyDCPM_V7_ACM.pdf

(Pathak et al., 2011)

A. Pathak,, Y.C. Hu, M. Zhang, P. Bahl, Y. Wang. Fine-grained power modeling for smartphones using system call tracing. Proceedings of the sixth conference on Computer systems. ACM. 2011. Retrieved 03/26/2011 from

http://dl.acm.org/citation.cfm?id=1966460

(Rambus, 2009)

Rambus. Challenges and Solutions for Future Main Memory. White Paper. May 26, 2009. Retrieved 03/26/2012 from

http://www.rambus.com/us/downloads/document_abstracts/products/future_main_memory_whitepaper.html

(Rivoire, 2008)

S. Rivoire. Models and Metrics for Energy-Efficient Computer Systems. Ph.D. Dissertation, June 2008. Department of Electrical Engineering, Stanford University. Retrieved 03/26/2012 from

http://csl.stanford.edu/~christos/publications/2008.suzy_rivoire.phd_thesis.pdf

(Rivoire et al., 2007)

S. Rivoire, M.A. Shah, P. Ranganatban, C. Kozyrakis, J. Meza. Models and Metrics to Enable Energy-Efficiency Optimizations. Computer. Vol. 40, Issue 12, pp. 39-48. December 2007. Retrieved 03/26/2012 from

http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4404808

(Rusu et al., 2006)

C. Rusu, A. Ferriera, C. Scordino, A. Watson, R. Melhem, D. Mosse. Energy Efficient Real-Time Heterogeneous Server Clusters. Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium, pp. 418-428, 2006. IEEE Computer Society. Retrieved 03/26/2012 from

http://dl.acm.org/citation.cfm?id=1128459

(Sawyer, 2011)

R. Sawyer. Calculating Total Power Requirements for Data Centers. White Paper 3, Revision 1. Schneider Electric white paper library. Schneider Electric Data Center Science Center. 2011. Retrieved 03/26/2012 from

http://www.apcmedia.com/salestools/VAVR-5TDTEF_R1_EN.pdf

(Sharma et al., 2006)

S. Sharma, C. Hsu, W. Feng. Making a Case for a Green500 List. Proceedings of the Workshop on High-Performance, Power-Aware Computing, 2006. Retrieved 03/26/2012 from

http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1639600

(Shye et al., 2009)

A. Shye, B. Scholbrock, G. Memik. Into the wild: studying real user activity patterns to guide power optimizations for mobile architectures. In Proc. of MICRO, 2009. Retrieved 03/26/2012 from

http://dl.acm.org/citation.cfm?id=1669135

(Song et al., 2009)

S. Song, R. Ge, X. Feng, K. W. Cameron. Energy Profiling and Analysis of the HPC Challenge Benchmarks. IJHPCA Vol. 23, No. 3, pp 265-276. Fall 2009. Retrieved 03/26/2012 from

http://hpc.sagepub.com/content/23/3/265.short

(SPEC, 2009)

SPEC Power and Performance, Benchmark Methodology V1.1.1. Standard Performance Evaluation Corporation. 04/01/2009.

(SPEC, 2011)

SPEC Power and Performance, Benchmark Methodology V2.1. Standard Performance Evaluation Corporation. 08/17/2011. Retrieved 03/26/2012 from

http://www.spec.org/power/docs/SPEC-Power_and_Performance_Methodology.pdf

(TDP, 2004)

M. Chin. Athlon 64 for Quite Power. Silent PC Review. 06/15/2004. Retrieved 03/26/2012 from

http://www.silentpcreview.com/article169-page3.html

(TDP, 2011)

Thermal Design Power. CPU World. March, 2011.Retrieved 03/26/2012 from http://www.cpu-world.com/Glossary/T/Thermal_Design_Power_(TDP).html

(Top500, 2008)

Power consumption of Supercomputers. November 2008. Retrieved 03/26/2012 from http://www.top500.org/lists/2008/06/highlights/power

(Tyan, 2003)

S2720-533 Thunder i7501 User's Manual. Tyan Computer Corporation, Revision 1. 2002-2003.

(WU, 2008)

watts up? Pro Operators Manual. Electronic Educational Devices Incorporated. 02/08 Rev. 9.

(Wang et al., 2010)

X. Wang, X. Fu, X. Liu, Z. Gu. PAUC: Power-Aware Utilization Control in Distributed Real-Time Systems. IEEE Transactions on Industrial Informatics. Vol. 6, Issue 3, pp. 302 – 315. Aug. 2010. Retrieved 03/26/2012 from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5484607

(Zeng et al., 2002)

Heng Zeng, Carla S. Ellis, Alvin R. Lebeck, Amin Vahdat. Ecosystem: Managing energy as a first class operating system resource. In Proc. of ASPLOS, 2002. Retrieved 03/26/2012 from http://dl.acm.org/citation.cfm?id=605411

(Zhang et al., 2010)

L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R.P. Dick, Z.M. Mao, L. Yang. Accurate Online Power Estimation and Automatic Battery Behavior Based Power Model Generation for Smartphones. In Proc. of CODES+ISSS, 2010. Retrieved 03/26/2012 from http://dl.acm.org/citation.cfm?id=1878982

Hardware Performance Counters

(Kufrin, 2005)

R. Kufrin. Measuring and Improving Application Performance with PerfSuite. Linux Journal, Issue #135, July 2005. Retrieved 03/26/2012 from

http://perfsuite.ncsa.illinois.edu/publications/LJ135/t1.html

(LM, 2012)

How to Install and use lm_sensors. Linux Appliance Design Website. Retrieved 03/26/2012 from

http://www.runtimeaccess.com/articles/sensors/sensors.html

(LM Doc, 2012)

lm_sensors Documentation. Hardware Monitoring by lm-sensors. Retrieved 03/26/2012 from

http://www.lm-sensors.org/wiki/Documentation

(London et al., 2001)

K. London, S. Moore, P. Mucci, K. Seymour, R. Luczak. The PAPI Cross-Platform Interface to Hardware Performance Counters. Department of Defense Users' Group Conference Proceedings, pp. 18-21, 2001. Retrieved 03/26/2012 from

http://icl.cs.utk.edu/publications/pub-papers/2001/papi-ugc2001.pdf

(Mucci et al., 1999)

P.J. Mucci, S. Browne, C. Deane, G. Ho. PAPI: A Portable Interface to Hardware Performance Counters. Proceedings of the Department of Defense HPCMP Users Group Conference. pp. 7-10, 1999. Retrieved 03/26/2012 from

http://web.eecs.utk.edu/~mucci/latest/pubs/dodugc99-papi.pdf

(PAPI, 2011)

PAPI Online Reference Manual. October 2011. Retrieved 03/26/2012 from

http://icl.cs.utk.edu/projects/papi/wiki/Main_Page

(PAPIC, 2012)

Introduction to PAPI-C. PAPI. Retrieved 03/26/2012 from

http://icl.cs.utk.edu/projects/papi/wiki/Introduction_to_PAPI-C

(PAPIC CDI, 2012)

PAPIC:CDI. Component PAPI CDI: Component Development Interface. PAPI. Retrieved 03/26/2012 from

http://icl.cs.utk.edu/projects/papi/wiki/PAPIC:CDI#PAPI-C_Open_Research_Issues

(perfmon2, 2007)

perfmon2, The Hardware-based Performance Monitoring Interface for Linux. OpenSource.hp.com. 10/18/2007. Retrieved 03/26/2012 from

http://perfmon2.sourceforge.net/pfmon_usersguide.html

The /proc file system

(Birnbaum, 2005)

J. Birnbaum. The Linux /proc Filesystem as a Programmer's Tool. Software. June 17, 2005. Retrieved 03/26/2012from

http://www.linuxjournal.com/article/8381

(Blackfin, 2012)

The Proc File System: Introduction. Analog Devices, Inc. 2010-2012. Retrieved 03/26/2012 from

http://docs.blackfin.uclinux.org/doku.php?id=linux-kernel:proc_file_system

(Blakey, 2006)

J. Blakey. Overview of Linux Memory Management Concepts: Slabs. St. Thomas, USVI. 2006. Retrieved 03/26/2012

http://www.secretmango.com/jimb/Whitepapers/slabs/slab.html

(Bowden and Bauer et al., 2009)

T. Bowden, B. Bauer, J. Nerin, S. Feng. S. Seibold. The /proc Filesystem. /usr/src/linux-2.6.39/Documentation/filesystems/proc.txt. Retrieved 03/26/2012 from

virgo2.ece.utep.edu

(Grover, 2004)

S. Grover. Understanding the proc file system. 2004. Retrieved 03/26/2012 from

http://www.linuxfocus.org/English/January2004/article324.shtml

(iostats, 2012)

ricklind@us.ibm.com. I/O statistics fields. /usr/src/linux-2.6.39/Documentation/iostats.txt. Retrieved 03/26/2012 from

virgo2.ece.utep.edu

(Jones, 2006)

M.T. Jones. Access the Linux kernel using the /proc filesystem. March 14, 2006. Retrieved 03/26/2012 from

http://www.ibm.com/developerworks/linux/library/l-proc.html

(Jones, 2007)

M.T. Jones. Anatomy fo the Linux slab allocator. May 15, 2007. Retrieved 03/26/2012 from

http://www.ibm.com/developerworks/linux/library/l-linux-slab-allocator/

(Kay, 2011)

T. Kay. Linux Swap Space. Linux Journal. March 1, 2011. Retrieved 03/26/2012 from

http://www.linuxjournal.com/article/10678

(Kleen, 2005)

A. Kleen. Add 4GB DMA32 zone. LWN.net. September 2005. Retrieved 03/26/2012 from

http://lwn.net/Articles/152337/

(Lameter, 2006)

C. Lameter. Light weight event counters V4. Email communication from clameter@sgi.com to akpm@osdl.org. LWN.net. June 2006. Retrieved 03/26/2012 from

http://lwn.net/Articles/188327/

(LHT, 2011)

/proc/stat explained. 2011. Retrieved 03/26/2012 from

http://www.linuxhowtos.org/System/procstat.htm

(Lind, 2010)

R. Lind. Linux Scheduler Statistics, version 12. 02/15/2010. Retrieved 03/26/2012 from

http://eaglet.rain.com/rick/linux/schedstats/v12/format-12.html

(LPM, 2005)

R. van Riel. proc - process information pseudo-filesystem. Linux Programmer's Manual. proc(5) manual page. May 12, 2005. Retrieved 03/26/2012 from

virgo2.ece.utep.edu

(LPM, 2010)

proc - process information pseudo-file system. Linux Programmer's Manual. October 30, 2010. Retrieved 03/26/2012 from

http://www.kernel.org/doc/man-pages/online/pages/man5/proc.5.html

(Mouw, 2001)

E. (J.A.K.) Mouw. Linux Kernel Procfs Guide. Deft University of Technology. Revision 1.0. 2001. Retrieved 03/26/2012 from

http://www.stillhq.com/pdfdb/000445/data.pdf

(Nguyen, 2004)

B. Nguyen. Linux Filesystem Hierarchy. Version 0.65. 07/30/2004. Retrieved 03/26/2012 from

http://tldp.org/LDP/Linux-Filesystem-Hierarchy/html/proc.html

(nmi, 2004)

Non-Maskable Interrupts (NMI). PCGuide. Retrieved 03/26/2012 from http://www.pcguide.com/ref/mbsys/res/irq/funcNMI-c.html

(Redhat, 2003)

Tips & Tricks. /proc/meminfo Explained. March 2003. Retrieved 03/26/2012 from http://www.redhat.com/advice/tips/meminfo.html

(RHEL 5, 2012)

Redhat Enterprise Linux 5, Deployment Guide. Deployment, Configuration, and Administration of Red Hat Enterprise Linux 5. Edition 8. Redhat Inc., 2012. Retrieved 03/26/2012 from http://docs.redhat.com/docs/en-US/Red_Hat_Enterprise_Linux/5/html/Deployment_Guide/ch-proc.html

(RHEL 6, 2011)

Redhat Enterprise Linux 6, Deployment Guide. Deployment, Configuration, and Administration of Red Hat Enterprise Linux 6. Edition 2. Redhat Inc., 2011. Retrieved 03/26/2012 from http://docs.redhat.com/docs/en-US/Red_Hat_Enterprise_Linux/6/html/Deployment_Guide/index.html

(Salzman et al., 2005)

P.J. Salzman, M. Burian, O. Pomerantz. The Linux Kernel Module Programming Guide. Chapter 5. The /proc File System. Ver. 2.6.4, 05/18/2005. Retrieved 03/26/2012 from http://tldp.org/LDP/lkmpg/2.6/html/x710.html

(Sigoure, 2011)

B. Sigoure. Clarifications on Linux's NUMA stats. Tsuna's blog. June 3, 2011. Retrieved 03/26/2012 from http://blog.tsunanet.net/2011/06/clarifications-on-linuxs-numa-stats.html

(Smith, 2008)

M.H. Smith. Redhat Enterprise Linux, Deployment Guide, Revision 5.2-5. Redhat Inc., 2008. Retrieved 03/26/2012 from http://www.centos.org/docs/5/html/5.2/Deployment_Guide/ch-proc.html

(SourceForge, 2009)

Memory Fragmentation. Feb. 12, 2009. Retrieved 03/26/2012 from

http://collectl.sourceforge.net/BuddyInfo.html

(vmstat, 2006)

/proc/vmstat. The /proc filesystem documentation. LinuxInsight. 08/26/2006. Retrieved 03/26/2012 from

http://www.linuxinsight.com/proc_vmstat.html

(Wettstein, 1999)

G. Wettstein. klogd – Kernel Log daemon. Linux System Administration. klogd(8) manual page. Version 1.4. 1999. Retrieved 03/26/2012 from

virgo2.ece.utep.edu

(zoneinfo, 2011)

Check usage of different parts of memory. 5/10/11.Retrieved 03/26/2012 from

http://stackoverflow.com/questions/5951718/check-usage-of-different-parts-of-memory

virgo2, Rocks, MPI, HPCC

(Gumataotao, 2010)

N.C. Gumataotao. Email Communication. 2010.

(hpcc-faq, 2012)

HPCC FAQ. Retrieved 03/26/2012 from

http://icl.cs.utk.edu/hpcc/faq/index_print.html

(Luszczek et al., 2005)

P. Luszczek, J. Dongarra, D. Koester, B. Rabenseifner, J. Lucas, J. Kepner, J. McCalpin, D. Bailey, D. Takahashi. *Introduction to the HPC Challenge Benchmark Suite.* Seattle, WA. March 2005. Retrieved 03/26/2012 from

http://icl.cs.utk.edu/projectsfiles/hpcc/pubs/hpcc-challenge-benchmark05.pdf

(OpenMPI, 2012)

OpenMPI FAQ: General run-time tuning. 02/21/12. Retrieved 03/26/2012 from http://www.open-mpi.org/faq/?category=tuning

(Rocks, 2011)

Rocks. Base users guide. University of California. 2011. Retrieved 03/26/2012 from http://www.rocksclusters.org/roll-documentation/base/5.4.3/

R

(CRAN, 2012)

CRAN. Contributed Packages. Retrieved 03/26/2012 from http://cran.cnr.berkeley.edu/

(R, 2012)

The R Project for Statistical Computing. r-project.org.

(rpanel, 2007)

A. Bowman, E. Crawford, G. Alexander, R.W. Bowman. rpanel: Simple Interactive Controls for R Functions Using the tcltk Package. Journal of Statistical Software, Volume 17, Issue 9. The American Statistical Association. January 2007. Retrieved 03/26/2012 from http://cran.r-project.org/web/packages/rpanel/rpanel.pdf

Coefficient of determination

(CoefD, 2004)

Coefficient of Determination. Course Notes. Retrieved 03/26/12from http://www.public.iastate.edu/~alicia/stat328/Regression%20inference-part3.pdf

(CoefD, 2009)

How to Find the Coefficient of Determination. Statistics How To. Retrieved 03/26/2012 from http://www.statisticshowto.com/articles/how-to-find-the-coefficient-of-determination/

(CoefD, 2012)

Correlation Coefficient. How well does your regression equation truly represent your set of data? Retrieved 03/26/2012 from

http://mathbits.com/mathbits/tisection/statistics2/correlation.htm

(Knight, 1980)

J.L. Knight. The Coefficient of Determination and Simultaneous Equation Systems. Journal of Econometrics 14, pp. 265-270. North-Holland Publishing Company. 1980. Retrieved 03/26/2012 from

http://www.sciencedirect.com/science/article/pii/0304407680900962

Factor Analysis

(Delic et al., 1996)

G. Delic, R.I. Haller. Factor Analysis of Applications Performance Data for the Cray Y-MP *International Journal of High Performance Computing Applications March 1996 10: 91-113*.

(Feser, 2005)

E. Feser. Benchmark value chain industry clusters for applied regional research. Department of Urban and Regional Planning & Regional Economics Applications Laboratory (REAL), University of Illinois at Urbana-Champaign. October 2005. Retrieved 03/26/2012 from

http://www.urban.illinois.edu/Faculty/feser/Pubs/VC_Methods.pdf

(Garson, 2012)

G.D. Garson. Factor Analysis. Statistical Associates Publishing, Blue Book Series. 2012. Retrieved 03/26/21012 from

http://www.statisticalassociates.com/factor.pdf

(Jain, 1991)

R.K. Jain. The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling. John Wiley and Sons, Inc. April, 1991.

(Kabacoff, 2012)

R.I. Kabacoff. Principle Components and Factor Analysis. Quick-R. 2012. Retrieved 03/26/2012 from

http://www.statmethods.net/advstats/factor.html

(Miles, 2005)

J.V.N. Miles. Confirmatory factor analysis using Microsoft Excel. Behavior Research Methods, 37(4), pp 672-676. Pschomomic Society, Inc. 2005. Retrieved 03/26/2012 from

http://www.jeremymiles.co.uk/mestuff/publications/p36.pdf

(R, cor)

Correlation, Variance and Covariance (Matrices). R Documentation. Retrieved 03/26/2012 from

http://stat.ethz.ch/R-manual/R-patched/library/stats/html/cor.html

(R, factanal)

Factor Analysis. R Documentation. Retrieved 03/26/2012 from

http://stat.ethz.ch/R-manual/R-patched/library/stats/html/factanal.html

(Rummel, 2012)

R.J. Rummel. Understanding Factor Analysis. Scanned from "Understanding Factor Analysis". The Journal of Conflict Resolution, pp 444-480. December, 1967. Retrieved 03/26/2012 from

http://www.hawaii.edu/powerkills/UFA.HTM

(Steiger, 2009)

J.H. Steiger. Exploratory Factor Analysis with R. 2009. Retrieved 03/26/2012 from

http://www.statpower.net/Content/312/R%20Stuff/Exploratory%20Factor%20Analysis%20with%20R.pdf

(Tucker and MacCallum, 1993)

L.R. Tucker, R.C. MacCallum. Exploratory Factor Analysis, A book manuscript by Ledyard Tucker and Robert MacCallum. 1997. Retrieved 03/26/2012 from

http://www.unc.edu/~rcm/book/factor.pdf

Glossary

Power (Watts, W) and Energy (kWh)

Power

The rate of energy consumption at a time instance.

Energy

The accumulated transmission rate of energy consumption.

Watts_total

Total power consumption, in Watts, per sample as measured by a *watts up? Pro* meter.

Watts_idle

Total power consumption, in Watts, of a system that is idling. An idling system is one that has booted up and is waiting for work.

Watts_delta

Watts_delta is the amount of power consumption that occurs above "idle Watts" as a result of the system doing work. *Watts_delta* can be computed as follows: *Watts_delta* = *Watts_total* – *Watts_idle*.

Phase

Idle Phase

- An idle phase is a device or component state characterized by an extended period of inactivity.
- There is a threshold, idle_t, such that extended periods of activity that measure above this threshold defines computational phases.

Cooldown Phase

- A cool down phase is a period of time marked by a change in state from active to inactive by which the component or device is idle, yet cooling down, that is dissipating heat above normal idle.

Sampling

Capture point

Point in time data is captured.

Watt window

Set of consecutive Watt samples modeled for a particular analysis.

Open MPI

Processor affinity

MPI affinity is a technique to improve performance of a benchmark by binding or locking processes and memory to a single CPU core. Open MPI affinity is specified using the command line option "--mca mpi_paffinity_alone 1" or by defining and specifying a rankfile in the *mpirun* command line.

Rankfile

A rank file is a file that specifies a mapping of mpi processes to processor cores. The rank file specifies a host node and slot list binding for each MPI process.

Least Squares Regression

Residual (r)

A residual (or error), *r*, is the difference between the actual value and predicted value. In terms of Watts, a residual is the difference between actual Watts and predicted or modeled Watts.

Squared residual (r^2 or r2)

A squared residual is the square of a residual. Note this is not to be confused with Pearson's correlation coefficient, which in this work is represented as "cor" or "CORR".

Sum of squared residuals (R2 or SSE)

The sum of all squared residuals for a given power model. Note the difference between R2 and R^2: R2 represents the sum of the squared residuals in this work whereas R^2 and CoefD are used to represent the Coefficient of Determination.

Average of the sum of squared residuals ($R2_{avg}$ or R2_sum_avg)

Average sum of squared residuals is the value R2 divided by the number of sample points for a given model

Sum of squares (S2 or SS)

The sum of squares in the context of Watt meter samples is the sum of the squares of the difference between actual power consumption and average power consumption over a set of power measurements. That is S2 = Sum $(W_i - W_{avg})^2$, where i ranges from 1..n, W_i is power measurement of sample i, W_{avg} is the average of power consumption across a Watt meter data set.

Average sum of squares ($S2_{avg}$ or S2_avg)

Average sum of squares is the value S2 divided by the number of sample points for a given model.

Selected variables

A set of variables applied to power models resulting from either manual selection of utilization counters or automatic selection of utilization counters using an algorithm implemented in R.

Fitted variables

A set of fitted variables corresponds to a least squares regression that takes a non-metered compute node system activity matrix (comprised of the fitted variable's values) and calculates "*fitted*" coefficients based on another compute node's power load vector or set of samples.

Statistics

Coefficient of Determination (R^2 or CoefD)

The Coefficient of Determination is 1 minus the ratio of R2 to S2 (1 – R2/S2). *CoefD* is close to one when a linear regression fits data better when compared to average of the data.

Factor Analysis

Factor analysis

"A data reduction technique that seeks to find the fewest common dimensions among a set of variables" (Feser, 2005).

Factor loading

Represent the correlations of variables with factors in factor analysis.

Appendix

FT3: Chapter 3 Tables and Figures

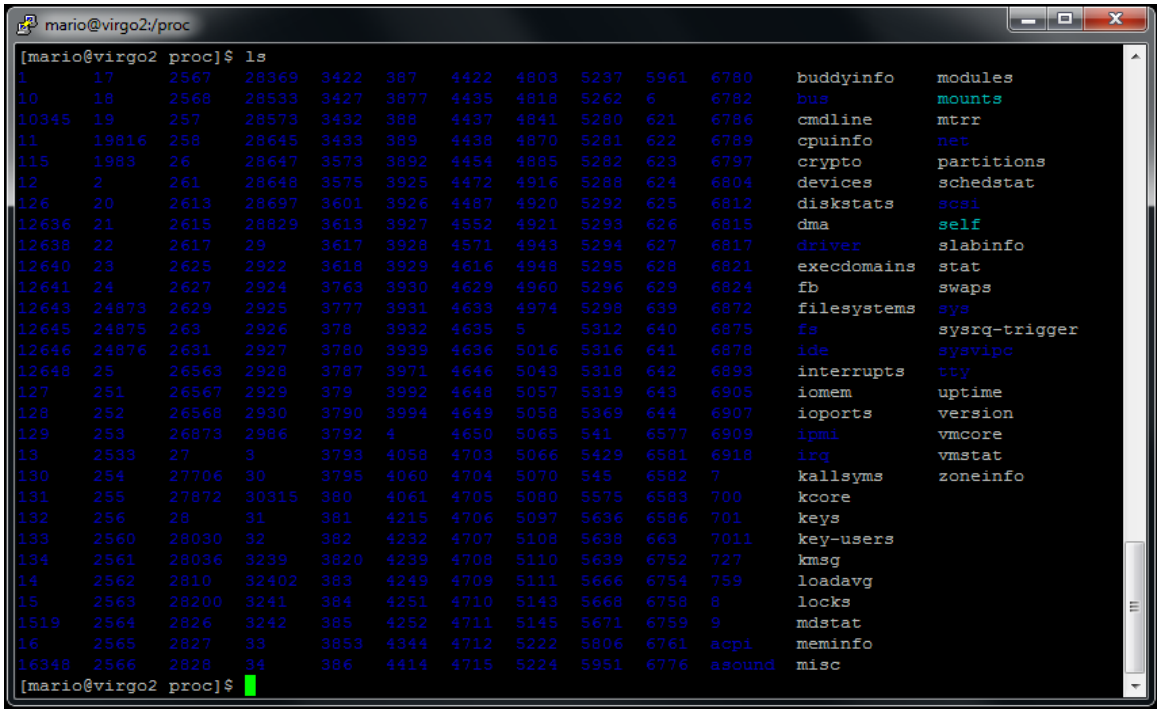


Figure 1 - Directory Listing of the /proc file system on *virgo2.ece.utep.edu*.

Table 1 – Non-numeric directories and files found in /proc on *virgo2.ece.utep.edu*.

acpi	execdomains	kcore	**mounts**	**sys**
buddyinfo	fb	keys	mtrr	sysrq-trigger
bus	filesystems	key-users	**net/dev**	**sysvipc**
cmdline	**fs**	kmsg	partitions	**tty**
cpuinfo	**ide**	**loadavg**	**schedstat**	uptime
crypto	**interrupts**	locks	**scsi**	version
devices	iomem	mdstat	**self**	vmcore
diskstats	ioports	**meminfo**	**slabinfo**	**vmstat**
dma	**irq**	misc	**stat**	**zoneinfo**
driver	kallsyms	modules	swaps	

Table 2 – Number of variables tracked per /proc file.

/proc file	Counters	Purpose
buddyinfo	33	Kernel buddy memory allocation
diskstats	385	Disk I/O
interrupts	138	Interrupt usage
loadavg	5	1, 5, 15 minute load averages
meminfo	30	Memory utilization and distribution
net/dev	80	Network interface counters
schedstat	672	Information on the Linux scheduler
slabinfo	1,216	Slab pool counters
stat	317	Overall system statistics
vmstat	51	Virtual memory statistics
TOTAL	**2,927**	*Total number of counters sampled per sample period*

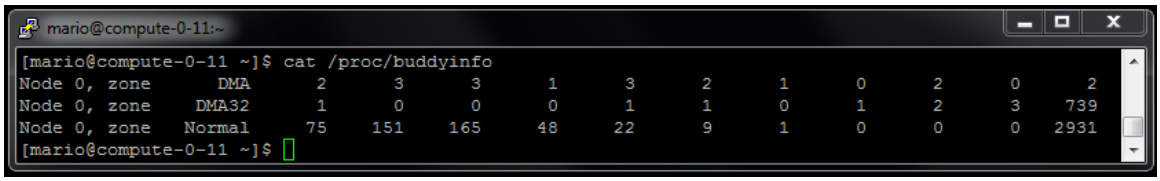


Figure 2 – Output from /proc/buddyinfo (*virgo2*, c0-11).

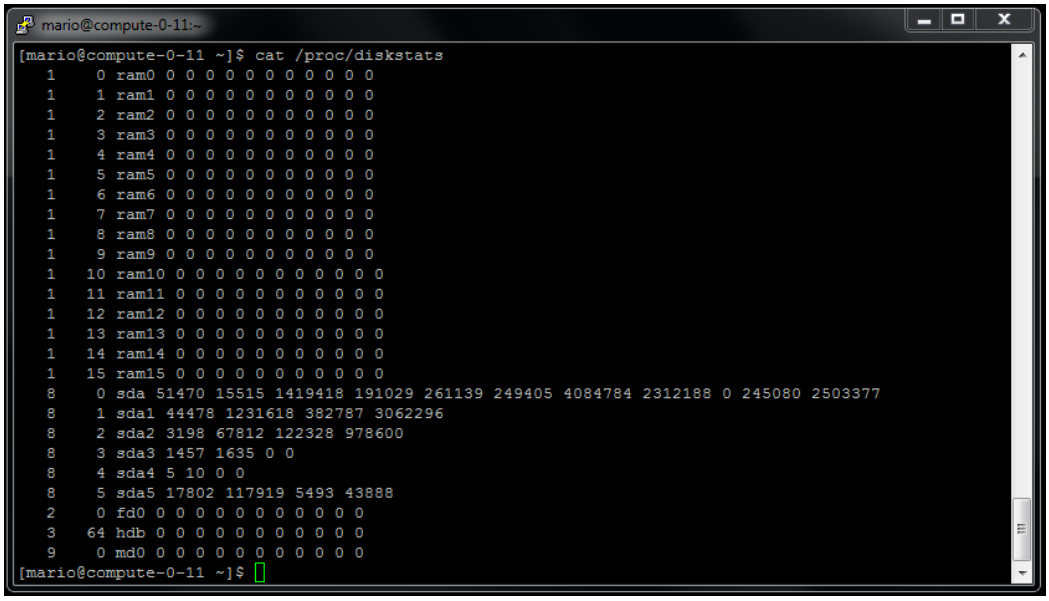


Figure 3 – Output from /proc/diskstats (*virgo2*, c0-11).

Table 3 – Description of /proc/diskstats *Disk* fields.

Variable	Field	Counter Description
num_r	1	# successfully completed reads
merged_r	2	# reads merged
sectors_r	3	# successfully read sectors
msec_r	4	# of milliseconds spent by all reads
complete_w	5	# successfully completed writes
merged_w	6	# writes merged
sectors_w	7	# successfully written sectors
msec_w	8	# milliseconds spent by all writes
num_io	9	# I/Os currently in progress
msec_io	10	# milliseconds spent doing I/O
msec_io_weighted	11	Weighted number of milliseconds spent doing I/O

Table 4 – Description of /proc/diskstats *Partition* fields.

Variable	Field	Counter Description
issued_r	1	# of issued reads
sectors_r	2	# of sectors requested to be read
issued_w	3	# of issued writes
sectors_w	4	# sectors requested to be written

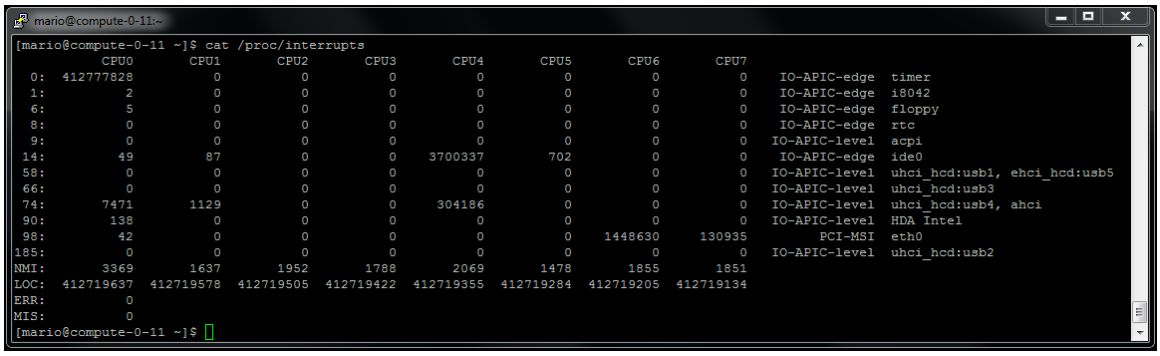


Figure 4 – Output from /proc/interrupts (*virgo2*, c0-11).

Table 5 - Description of IRQs associated with /proc/interrupts on *virgo2*.

IRQ	Type	Device Name
0	IO-APIC-edge	timer
1	IO-APIC-edge	i8042
6	IO-APIC-edge	floppy
7	IO-APIC-edge	parport0
8	IO-APIC-edge	rtc
9	IO-APIC-level	acpi
12	IO-APIC-edge	i8042
14	IO-APIC-edge	ide0
58	IO-APIC-level	uhci_hcd:usb1, ehci_hcd:usb5
66	IO-APIC-level	uhci_hcd:usb3
74	IO-APIC-level	uhci_hcd:usb4, ahci (or libata)
82	IO-APIC-level	eth2
90	IO-APIC-level	HDA Intel
98	PCI-MSI	eth0
122	PCI-MSI	hda_intel
138	PCI-MSI	eth0
154	PCI-MSI	eth1
185	IO-APIC-level	uhci_hcd:usb2

Figure 5 – Output from /proc/loadavg (*virgo2*, c0-11).

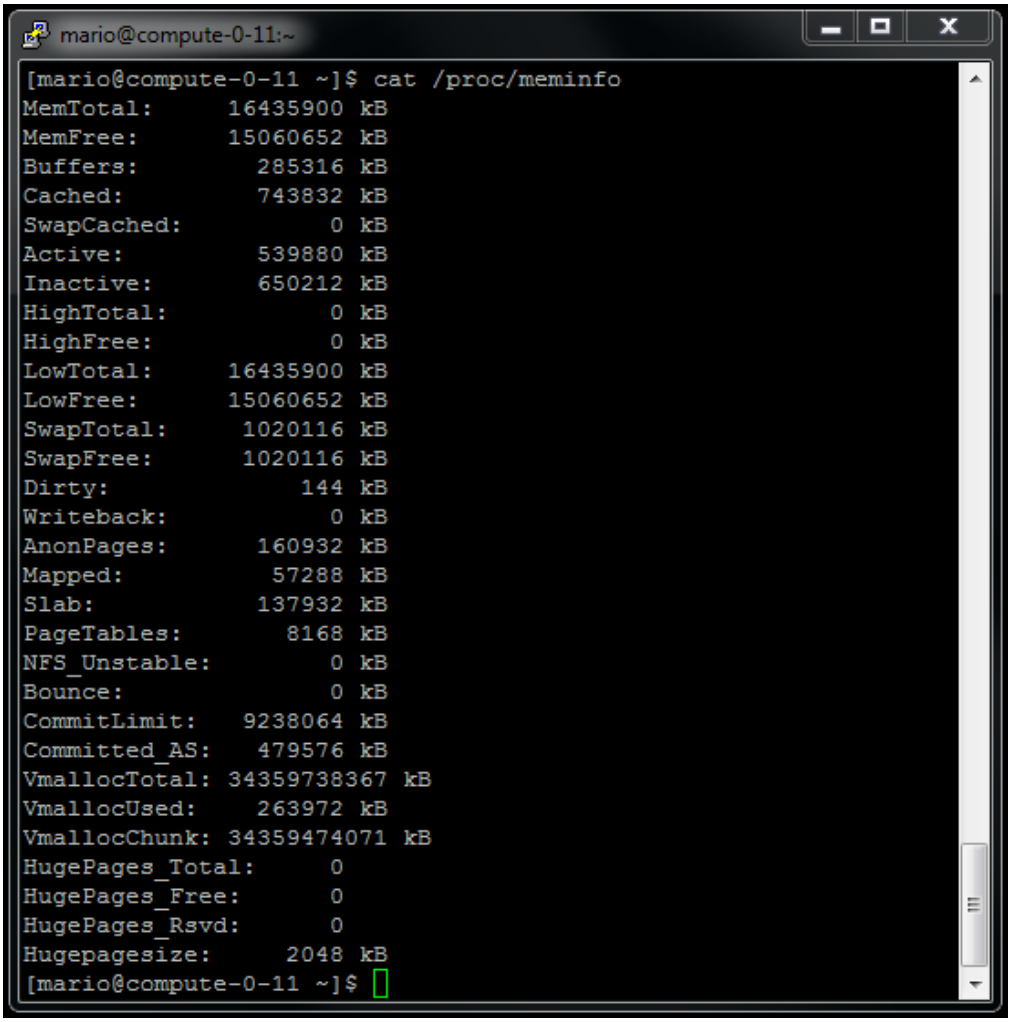


Figure 6 – Output from /proc/meminfo (*virgo2*, c0-11).

Table 6 – Description of /proc/meminfo fields.

Variable	Row #	Description
memF	2	Total low and high memory that is unused
cache	4	Amount of memory in page cache
active	6	Memory that has been used recently
inactive	7	Free and available
dirty	14	Total waiting to be written to disk
writeb	15	Total actively being written to disk
pagesA	16	Non file backed pages mapped into user space page tables
slab	18	Total used by kernel to cache objects (see slablinfo)
pageTables	19	Amount of memory dedicated to lowest page table level
commitA	23	Worst case RAM estimate to complete current workload

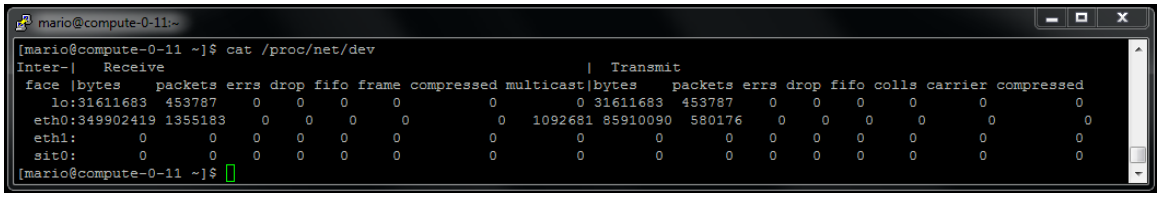


Figure 7 – Output from /proc/net/dev (*virgo2*, c0-11).

Table 7 – Listing of /proc/net/dev counters.

Column	**Receive**	**Transmit**
1	Bytes	Bytes
2	Packets	Packets
3	Errs	Errs
4	Drop	Drop
5	Fifo	Fifo
6	Frame	Rolls
7	Compressed	Carrier
8	Multicast	Compressed

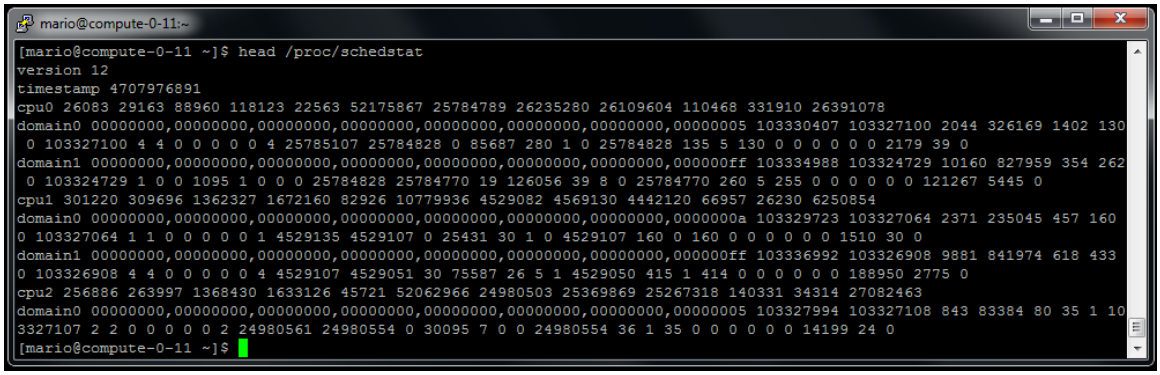


Figure 8 – First 10 lines of output from /proc/schedstat (*virgo2*, c0-11).

Table 8 – Description of /proc/schedstat version 12 CPU statistics.

Variable	Field	Counter Description
cpu_*core#*_0	1	# of times active and expired queue empty
cpu_*core#*_1	2	# of times active queue empty
cpu_*core#*_2	3	# of times expired queue empty
cpu_*core#*_3	4	# of times sched_yield() called
cpu_*core#*_4	5	**# of times the active queue has at least one other process on it**
cpu_*core#*_5	6	# of times schedule() called
cpu_*core#*_6	7	**# of times schedule() left processor idle**
cpu_*core#*_7	8	# of times try_to_wake_up() called
cpu_*core#*_8	9	# of times try_to_wake_up() process awakened last ran on waking cpu
cpu_*core#*_9	10	**sum of all time tasks spent running (in ms)**
cpu_*core#*_10	11	sum of all time tasks spent waiting (in ms)
cpu_*core#*_11	12	# of tasks (not necessarily unique) given to the processor

Table 9 – Description of /proc/schedstat version 12 domain statistics.

Variable	Field #	Counter Description
domain_*cpu#*_0	2	# of times load_balance() called when cpu idle
domain_cpu#_1	3	# of times load_balance() called when cpu idle, found no imbalance
domain_cpu#_2	4	# of times load_balance() called when cpu idle, failed to move one or more tasks
domain_cpu#_3	5	sum of imbalances discovered (if any) with each call to load_balance() when the cpu was idle
domain_cpu#_4	6	# of times load_balance() pulled tasks to this cpu while idle
domain_cpu#_5	7	# of times load_balance() pulled a task to this cpu even though it was hot on its original cpu
domain_cpu#_6	8	# of times load_balance() called but did not find a busier queue while the cpu was idle
domain_cpu#_7	9	# of times a busier queue was found while the cpu was idle but no busier group was found
domain_cpu#_8	10	# of times load_balance() called, cpu just becoming idle
domain_cpu#_9	11	# of times load_balance() called, cpu just becoming idle, found no imbalance
domain_cpu#_10	12	# of times load_balance() failed to move one or more tasks, cpu just becoming idle
domain_cpu#_11	13	sum of imbalances discovered (if any) with each call to load_balance() when the cpu was just becoming idle
domain_cpu#_12	14	# of times load_balance() pulled tasks to this cpu when newly idle
domain_cpu#_13	15	# oh times load_balance()pulled tasks to this cpu even thought the tasks were hot in their original cpus
domain_cpu#_14	16	# of times load_balance() called but did not find a busier queue while the cpu was just becoming idle
domain_cpu#_15	17	# of times a busier queue was found while cpu just becoming idle, no busier group found
domain_cpu#_16	18	# of times load_balance() called when cpu busy
domain_cpu#_17	19	# of times load_balance() called when cpu busy, found no imbalance
domain_cpu#_18	20	# of times load_balance() failed to move one or more tasks, cpu busy
domain_cpu#_19	21	sum of imbalances discovered (if any) with each call to load_balance(), cpu busy
domain_cpu#_20	22	# of times load_balance() pulled tasks to this cpu while busy
domain_cpu#_21	23	# of times load_balance() called when cpu busy, failed to find busier queue
domain_cpu#_22	24	# of times a busier queue was found while the cpu was busy but no busier group was found
domain_cpu#_23	25	# of times active_load_balance() called
domain_cpu#_24	26	# of times active_load_balance() failed to move a task
domain_cpu#_25	27	# of times active_load_balance() successfully pushed a task off a processor
domain_cpu#_32	34	# of times try_to_wake_up() woke a process on a remote cpu
domain_cpu#_33	35	# of times try_to_wake_up() decided to choose a cpu based on SD_WAKE_AFFINE.
domain_cpu#_34	36	# of times try_to_wake_up() decided to choose a cpu based on SD_WAKE_BALANCE.

Note: domain_*cpu#*_26 through 33 are always 0

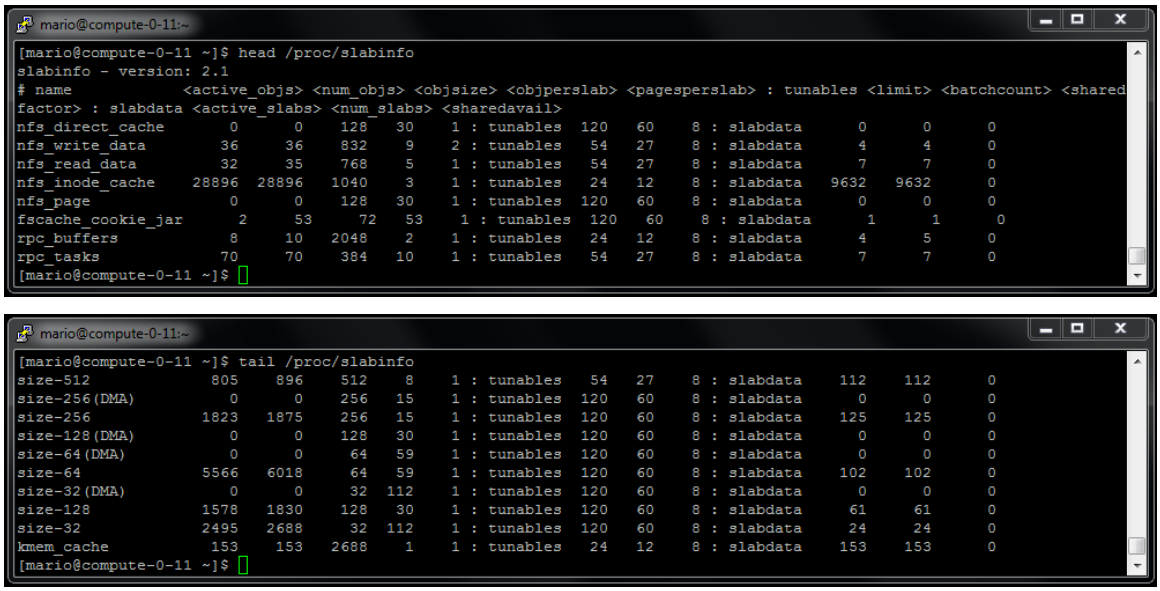


Figure 9 – Output from /proc/slabinfo (*virgo2*, c0-11).

Table 10 – Description of /proc/slabinfo columns.

Column Label	Column #	Description
active_objs	1	# of active objects
num_objs	2	# of objects
objsize	3	Object size
objperslab	4	# objects per slab
pagesperslab	5	Pages per slab
limit	6	Max # of objects cached per CPU (tunable)
batchcount	7	Max # of objects transferred to per-CPU cache (tunable)
sharedfactor	8	Indicates sharing behavior for SMP systems (tunable)
active_slabs	9	# of active slabs per slab cache
num_slabs	10	# of slabs per slab cache
sharedavail	11	Amount of shared available per slab cache

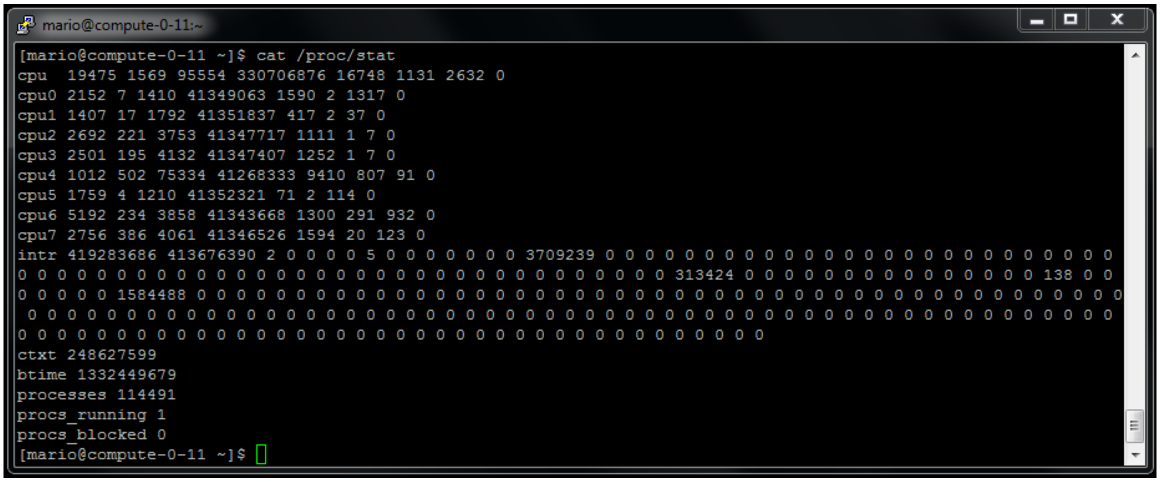


Figure 10 – Output from /proc/stat (*virgo2*, c0-11).

Table 11 – Description of CPU row fields found in /proc/stat.

Field No.	Field Name	Description
1	*cpu#*	CPU identifier
2	*user*	Time spent executing processes in user mode
3	*usern*	Time spent executing processes in user mode nice
4	*sys*	Time spent executing processes in system mode
5	*sysi*	Time spent idling
6	*iow*	Time spent waiting for I/O to complete
7	*hirq*	Time spent servicing hardware interrupts
8	*sirq*	Time spent servicing software interrupts
9	N/A	Always 0

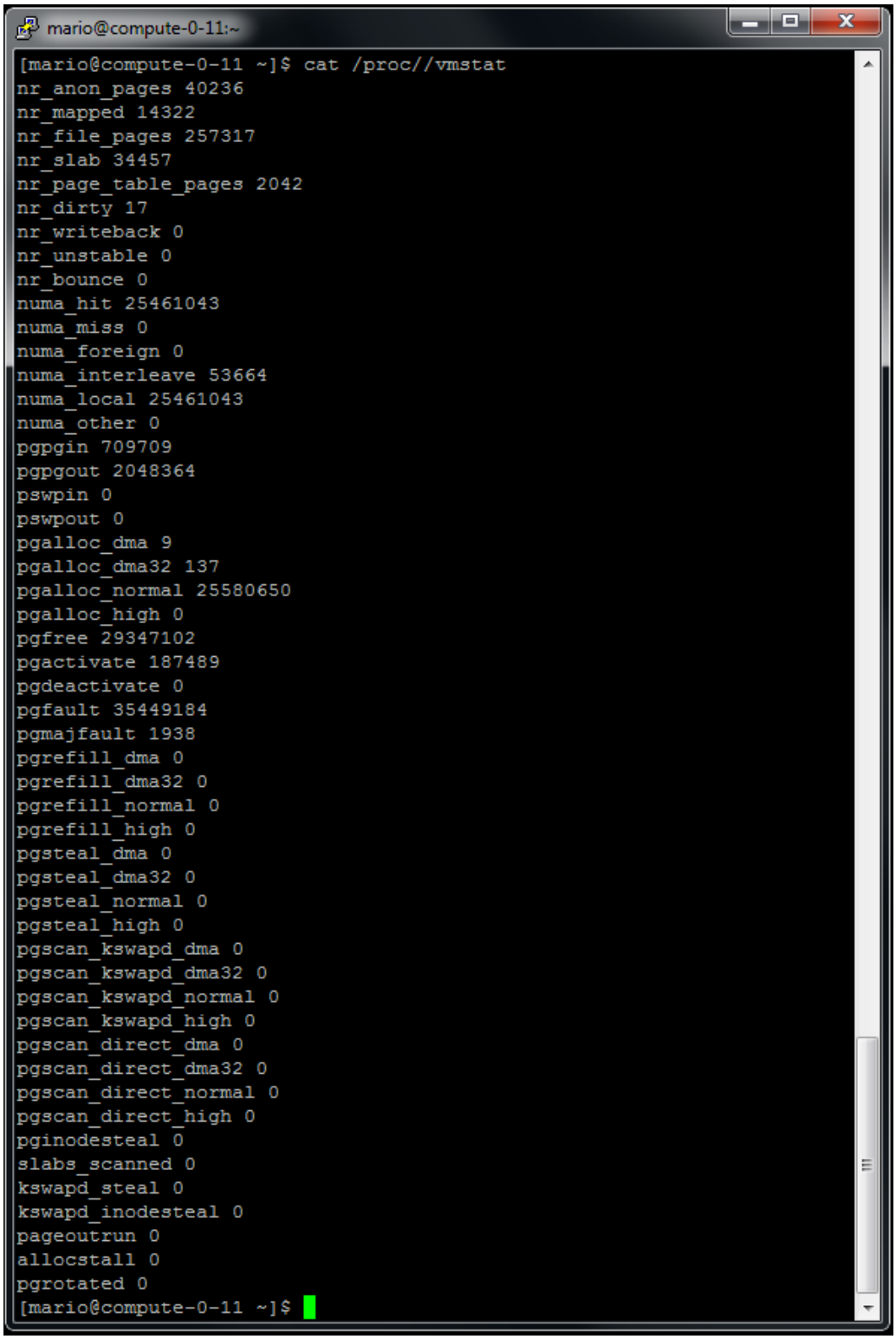


Figure 11 – Output from /proc/vmstat (*virgo2*, c0-11).

Table 12 – Description of /proc/vmstat target variables.

Row #	Variable	Description
1	*nr_anon_pages*	Mapped anonymous pages
4	*nr_slab*	# pages allocated by kernel slab allocator
5	*nr_page_table_pages*	# pages allocated to page tables
6	*nr_dirty*	# pages that are dirty
7	*nr_writeback*	# pages that are under writeback
10	*numa_hit*	Memory allocated in intended node
11	*numa_miss*	Memory allocated in non-intended node
16	*pgpgin*	# page ins (disk reads) since last boot
17	*pgpgout*	# page outs (disk writes) since last boot
20	*pgalloc_dma*	# pages allocated to dma zone
21	*pgalloc_dma32*	# pages allocated to dma32 zone
22	*pgalloc_normal*	# pages allocated to normal zone
23	*pgalloc_high*	# pages allocated to highmem zone
24	*pgfree*	# page frees since last boot
27	*pgfault*	# major + minor page faults since last boot
28	*pgmajfault*	# major page faults since last boot

Table 13 – Description of /proc/vmstat variables.

Row #	Variable	Description
1	*nr_anon_pages*	Mapped anonymous pages
2	*nr_mapped*	# pages mapped by files
3	*nr_file_pages*	# file backed pages or pagecache pages mapped into page tables.
4	*nr_slab*	# pages allocated by kernel slab allocator
5	*nr_page_table_pages*	# pages allocated to page tables
6	*nr_dirty*	# pages that are dirty
7	*nr_writeback*	# pages that are under writeback
8	*nr_unstable*	# network file system pages that are unstable
9	*nr_bounce*	# pages for bounce buffers
10	*numa_hit*	process tries to allocate memory this node, succeeded
11	*numa_miss*	process tries to allocate from another node, gets mem from this one
12	*numa_foreign*	Process tries to allocate on this node, gets mem from another
13	*numa_interleave*	Interleave attempt to allocate this node, succeeded
14	*numa_local*	process ran on this node, allocated memory on this node
15	*numa_other*	process ran on another node, allocated memory from other node
16	*pgpgin*	# page ins (disk reads) since last boot
17	*pgpgout*	# page outs (disk writes) since last boot
18	*pswpin*	# swap ins (swap reads) since last boot
19	*pswpout*	# swap outs (swap writes) since last boot
20	*pgalloc_dma*	# pages allocated to dma zone
21	*pgalloc_dma32*	# pages allocated to dma32 zone
22	*pgalloc_normal*	# pages allocated to normal zone
23	*pgalloc_high*	# pages allocated to highmem zone
24	*pgfree*	# page frees since last boot
25	*pgactivate*	# page activations since last boot (from inactive to active)
26	*pgdeactivate*	# page deactivations since last boot (from active to inactive)
27	*pgfault*	# major + minor page faults since last boot
28	*pgmajfault*	# major page faults since last boot
29	*pgrefill_dma*	# dma page refills since last boot
30	*pgrefill_dma32*	# dma32 page refills since last boot
31	*pgrefill_normal*	# normal page refills since last boot
32	*pgrefill_high*	# highmem page refills since last boot
33	*pgsteal_dma*	# dma page steals since last boot
34	*pgsteal_dma32*	# dma32 page steals since last boot
35	*pgsteal_normal*	# normal page steals since last boot
36	*pgsteal_high*	# highmem page steals since last boot
37	*pgscan_kswapd_dma*	# dma pages scanned by kswapd since last boot
38	*pgscan_kswapd_dma32*	# dma32 pages scanned by kswapd since last boot
39	*pgscan_kswapd_normal*	# normal pages scanned by kswapd since last boot
40	*pgscan_kswapd_high*	# highmem pages scanned by kswapd since last boot
41	*pgscan_direct_dma*	# dma pages reclaimed directly since last boot
42	*pgscan_direct_dma32*	# dma32 pages reclaimed directly since last boot
43	*pgscan_direct_normal*	# normal pages reclaimed directly since last boot
44	*pgscan_direct_high*	# normal pages reclaimed directly since last boot
45	*pginodesteal*	# pages reclaimed via inode freeing since last boot
46	*slabs_scanned*	# scanned slab objects since last reboot
47	*kswapd_steal*	# pages reclaimed by kswapd since last boot
48	*kswapd_inodesteal*	# pages reclaimed by kswapd via inode freeing since last boot
49	*pageoutrun*	# calls to page reclaim by kswapd since last boot
50	*allocstall*	# direct page reclaims since last boot
51	*pgrotated*	# pages rotated to least recently used (LRU)

Table 14 – List of *target* variables selected for power modeling.

stat	loadavg	meminfo	net_dev	diskstats
user	*load_avg_1*	*memF*	*lo0_Rx_Bt*	*sda_17_com_w*
sys	*load_avg_5*	*buf*	*lo0_Tx_B*	*sda_17_mer_w*
sysi	*load_avg_15*	*cache*	*lo0_Rx_Pkt*	*sda_17_sec_w*
iow	*kse_r*	*active*	*lo0_Tx_Pkt*	*sda_17_msec_w*
hirq	*kse_n*	*inactive*	*eth0_Rx_B*	*sda_17_num_io*
sirq		*iowF*	*eth0_Tx_B*	*sda_17_msec_io*
int1		*dirty*	*eth0_Rx_Pkt*	*sda_17_msec_io_w*
int2		*writeb*	*eth0_Tx_Pkt*	*sda1_18_sec_r*
int16		*pagesA*		*sda1_18_msec_r*
int76		*mapped*		*sda4_21_mer_r*
int100		*slab*		*sda4_21_sec_r*
ctxt		*pageTables*		
procs		*nfs*		
procs_r		*commitA*		
procs_b				

Table 15 – List of *target* variables selected for power modeling.

interrupts	schedstat	vmstat	buddyinfo	slabinfo
eth0_CPU7	*cpu0_4*	*nr_anon_pages*	*dma32_0*	*ano_vma__122_ao*
hda_intel_CPU1	*cpu1_4*	*nr_mapped*	*dma32_1*	*bio___86_ao*
hda_intel_CPU4	*cpu2_4*	*nr_file_pages*	*dma32_2*	*bio_1__85_ao*
hda_intel_CPU5	*cpu3_4*	*nr_slab*	*dma32_3*	*blk_req__79_ao*
ide0_CPU1	*cpu4_4*	*nr_page_table_pages*	*dma32_4*	*crq_poo__45_ao*
nmi_CPU0	*cpu5_4*	*nr_dirty*	*dma32_5*	*den_cac__107_ao*
nmi_CPU1	*cpu6_4*	*nr_writeback*	*dma32_6*	*ext_ino_3_33_ao*
nmi_CPU2	*cpu7_4*	*nr_unstable*	*dma32_7*	*fil___108_ao*
nmi_CPU3	*cpu0_6*	*numa_hit*	*normal_0*	*fil___108_no*
nmi_CPU4	*cpu1_6*	*numa_interleave*	*normal_1*	*jou_hea__36_ao*
nmi_CPU5	*cpu2_6*	*numa_local*	*normal_2*	*jou_hea__36_no*
nmi_CPU6	*cpu3_6*	*pgpgout*	*normal_3*	*scs_cmd_c_26_ao*
nmi_CPU7	*cpu4_6*	*pgalloc_dma32*	*normal_4*	*sgp_16__30_ao*
timer_CPU0	*cpu5_6*	*pgalloc_normal*	*normal_5*	*sgp_32__29_ao*
usb_1_5_CPU1	*cpu6_6*	*pgfree*	*normal_6*	*siz_102__141_ao*
usb_1_5_CPU4	*cpu7_6*	*pgactivate*	*normal_7*	*siz_102__141_no*
usb_4_CPU1	*cpu0_9*	*pgfault*		*siz_102__141_sa*
usb_4_CPU4	*cpu1_9*	*pgmajfault*		*siz_128__150_ao*
usb2_CPU6	*cpu2_9*			*siz_204__139_ao*
usb2_CPU7	*cpu3_9*			*siz_204__139_no*
	cpu4_9			*siz_204__139_sa*
	cpu5_9			*siz_512__143_ao*
	cpu6_9			*siz_512__143_no*
	cpu7_9			*siz_64__148_ao*
				siz_819__135_ao
				skb_fcl_c_90_ao
				skb_hea_c_91_ao
				skb_hea_c_91_sa
				uni___40_no
				vm_are_s_116_ao
				vm_are_s_116_no

FT4: Chapter 4 Tables and Figures

Table 16 – Virgo2 cluster specifications.

Type	Nodes	CPUs	Cores	cpu MHz	Memory	Cache Size
Frontend	1	2	8	2000	8 GB	6144 KB
Compute (2 metered)	18	2	8	2000	8 GB	6144 KB
GPU	1	2	8	2000	8 GB	6144 KB
Memory	1	2	8	2000	64 GB	6144 KB

```
FILE * proc_fd = fopen(proc_file, "r");
setvbuf(proc_fd, NULL, _IONBF, 0);
char proc_buf [MAX_BUF_SIZE];
while(1){
  rewind(proc_fd);
  nread  =  fread(&proc_buf,  sizeof(char),  MAX_BUF_SIZE,
proc_fd);
  sleep(T0_sec);
 }
```

Figure 12 – proc_sample.c main sampling loop.

Table 17 – Characteristics of the HPCC benchmarks.

Benchmark	Spatial	Temporal	Measures	Units
RA	low	low	Random updates to integer memory	GUP/s
PTRANS	high	low	Data transfer rate	GB/s
DGEMM	high	high	Floating point performance	FLOPS
STREAM	high	low	Memory Bandwidth	GB/s
FFTE	low	high	Floating point and memory transfer performance	FLOPS
L/BW	low	low	Latency and bandwidth of communication patterns	msec, MB/s
HPL	high	high	Floating point performance	FLOPS

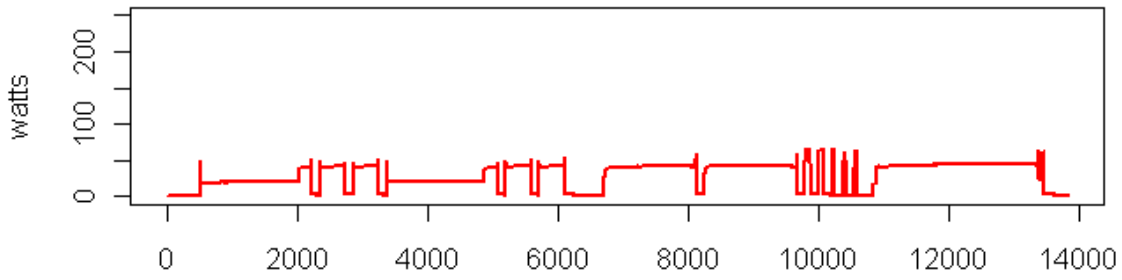


Figure 13 – Split HPCC benchmarks calculated on 1 core.

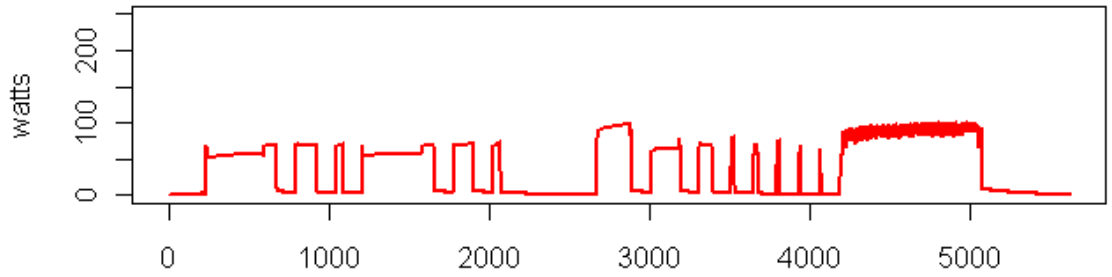


Figure 14 – Split HPCC benchmarks calculated on 4 cores.

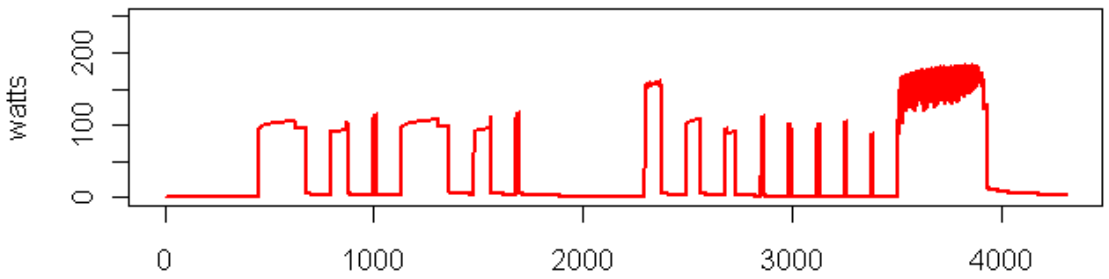


Figure 15 – Split HPCC benchmarks calculated on 8 cores.

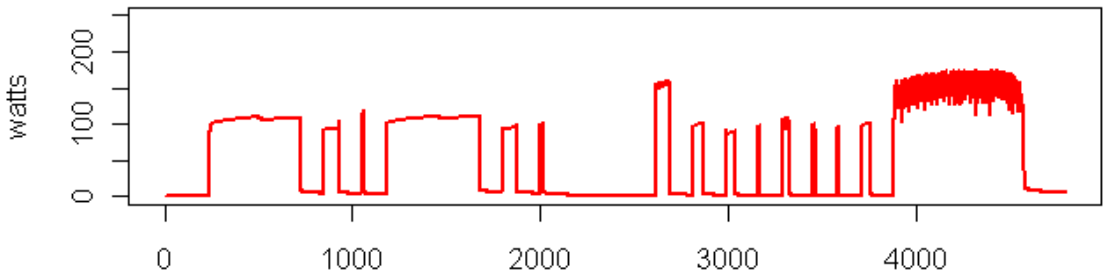


Figure 16 – Split HPCC benchmarks calculated on 16 cores (2 compute nodes).

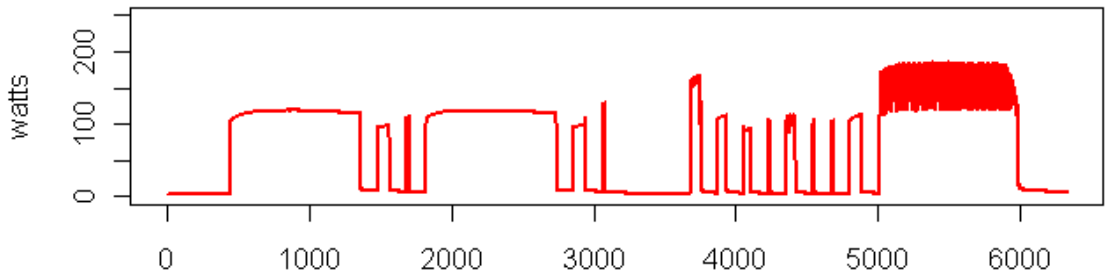


Figure 17 – Split HPCC benchmarks calculated on 32 cores (4 compute nodes).

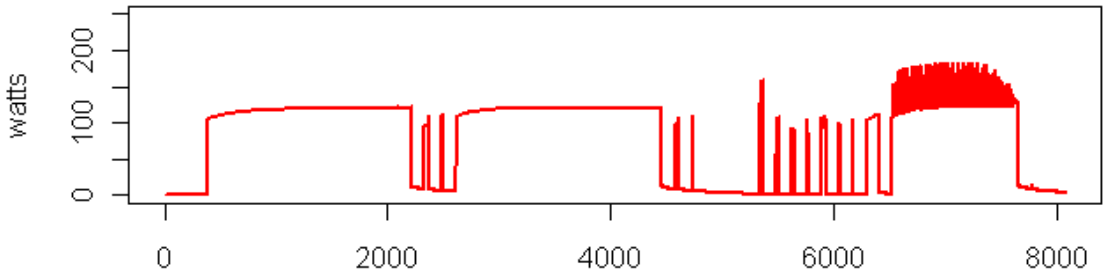


Figure 18 – Split HPCC benchmarks calculated on 160 cores (20 compute nodes).

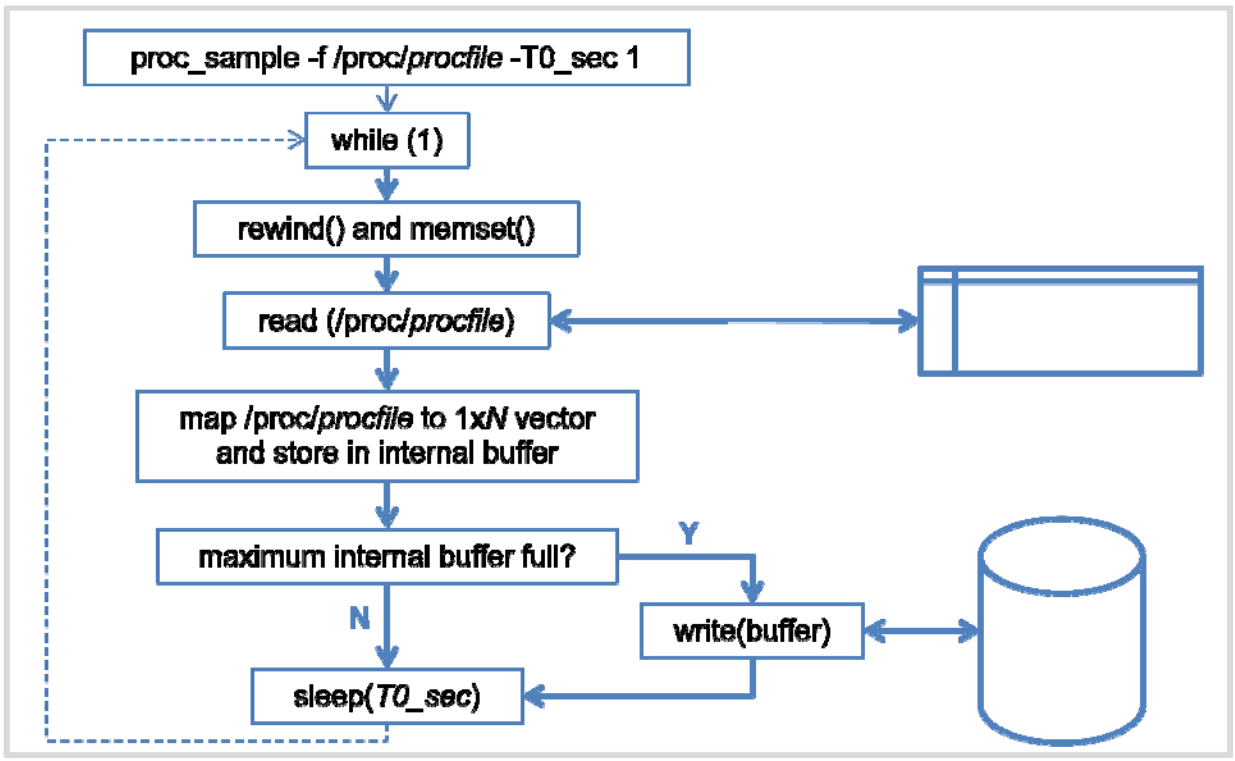


Figure 19 - Initial version of main loop for proc_sample.c.

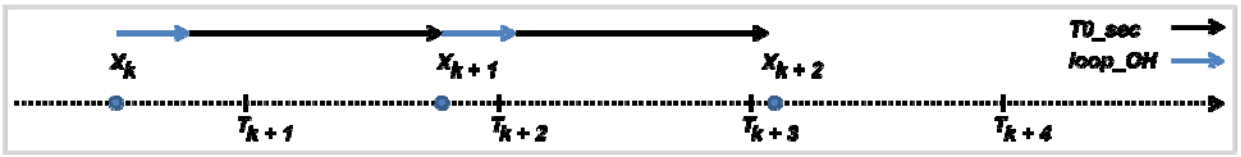


Figure 20 - Skipped sample in sampling interval $\{t_{k+2}, t_{k+3}\}$.

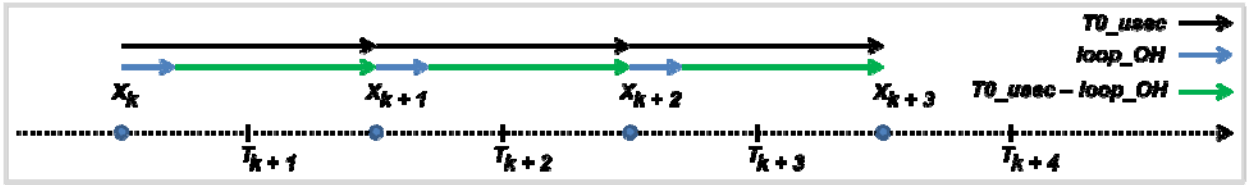


Figure 21 – Ideal sampling.

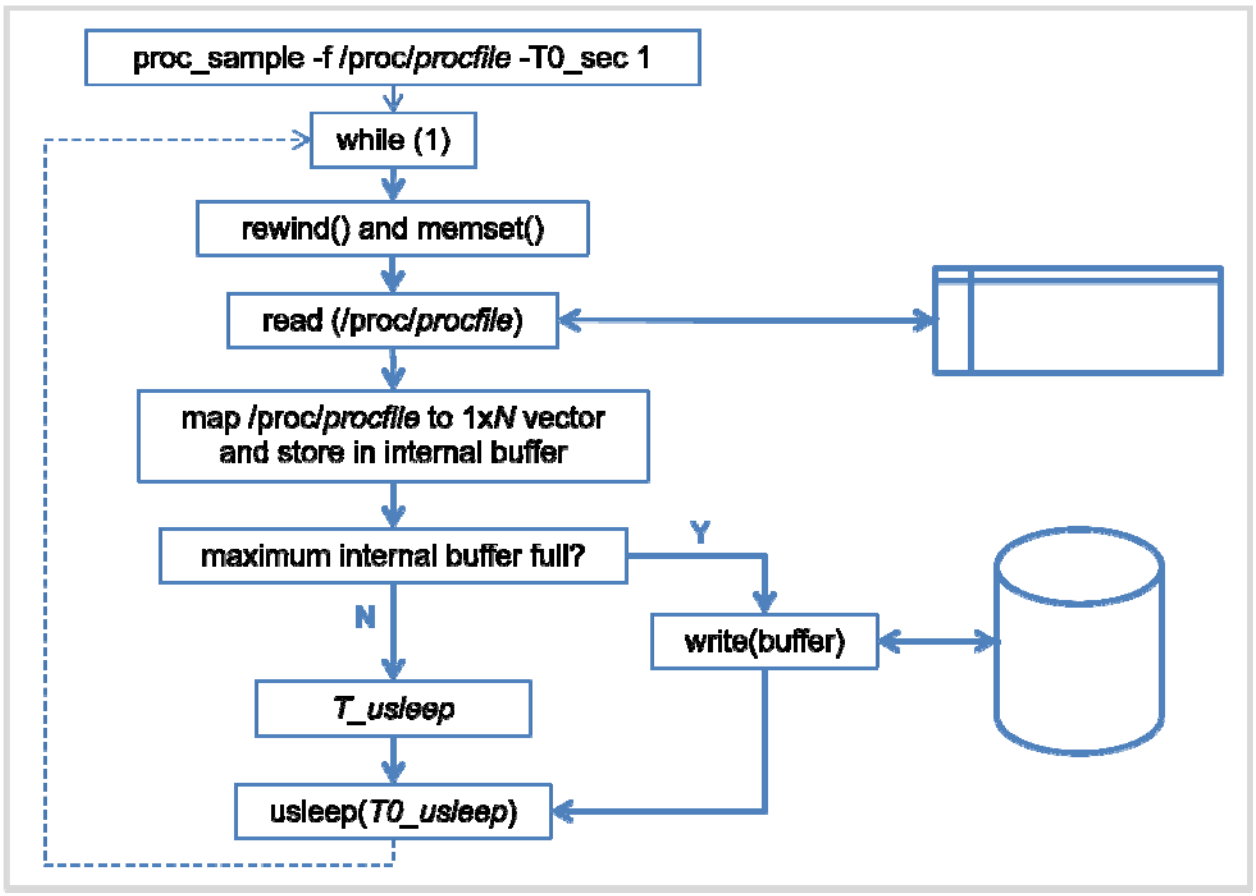


Figure 22 – Final sampling loop.

```
gettimeofday( &tv1, NULL);
if ( tv1.tv_usec < 499900 )
  usleep( 499900 - tv1.tv_usec );
```

Figure 23 – Wait for the midpoint of the sample period or immediately proceed.

```
gettimeofday( & Time_xk, NULL);
if (tv2.tv_usec > 500000)
  T0_usleep = T0_usec - (tv2.tv_usec - 500000);
else
  T0_usleep = T0_usec;
```

Figure 24 – Estimate the time when the next sample should be taken.

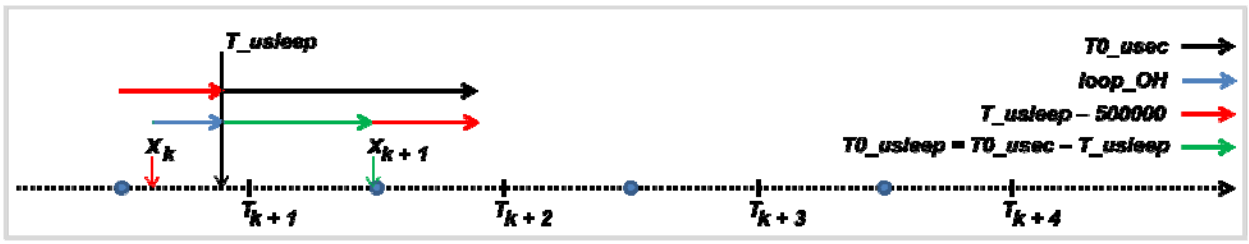


Figure 25 – Determining the sleep duration of the sampling process.

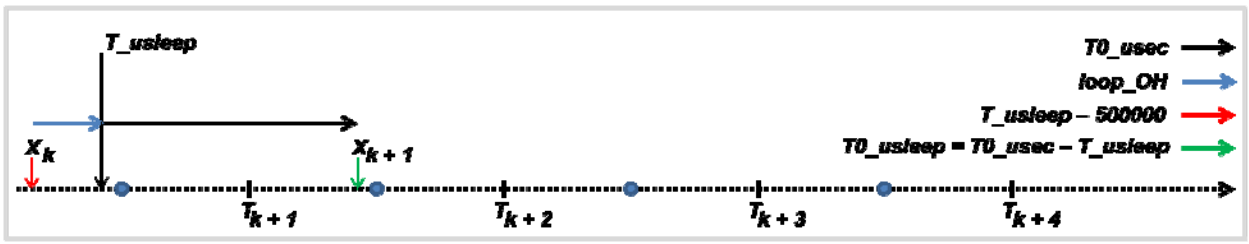


Figure 26 – Determining the sleep duration of the sampling process.

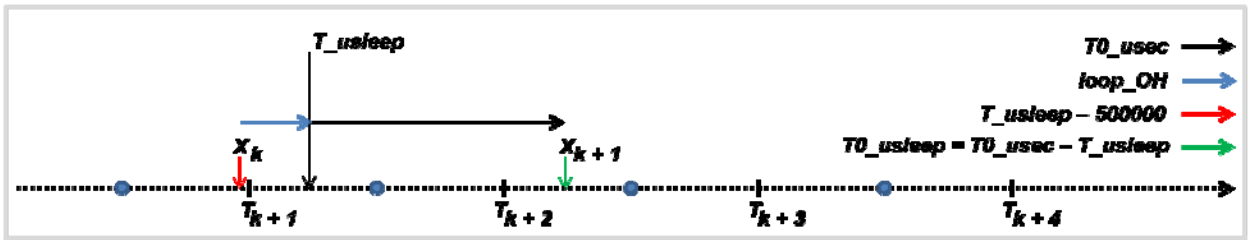


Figure 27 – Worst case scenario where a sample is skipped.

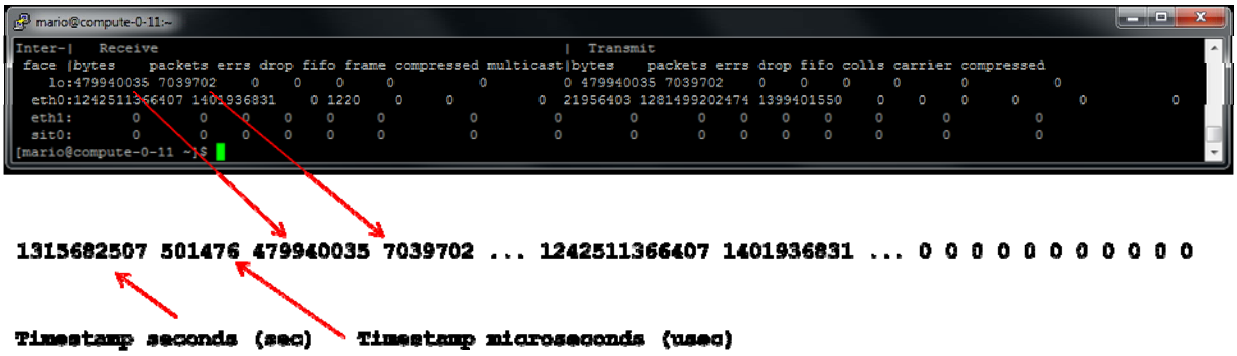


Figure 28 – Mapping /proc/net/dev to a 1 × *N* vector with timestamps.

Table 18 – Sample log file showing /proc/net/dev samples.

		lo		eth0		eth1		sit0	
sec	usec	Rx Bytes		Rx Bytes		Rx Bytes		Rx Bytes	
1315682507	501092	73894801	…	1234808192764	…	0	…	0	…
1315682508	501021	73894801	…	1234808192764	…	0	…	0	…
1315682509	502077	73894957	…	1234808247046	…	0	…	0	…
1315682510	501051	73894957	…	1234808247046	…	0	…	0	…
1315682511	501117	73895123	…	1234808248724	…	0	…	0	…
1315682512	501172	73895123	…	1234808248724	…	0	…	0	…
1315682513	501231	73895123	…	1234808250308	…	0	…	0	…
1315682514	501380	73895123	…	1234808250308	…	0	…	0	…
1315682515	501352	73895123	…	1234808250308	…	0	…	0	…
1315682516	501418	73895289	…	1234808250308	…	0	…	0	…
1315682517	500471	73895289	…	1234808250308	…	0	…	0	…
1315682518	500533	73895289	…	1234808250308	…	0	…	0	…
1315682519	500592	73895289	…	1234808250308	…	0	…	0	…
1315682520	500653	73895455	…	1234808250308	…	0	…	0	…

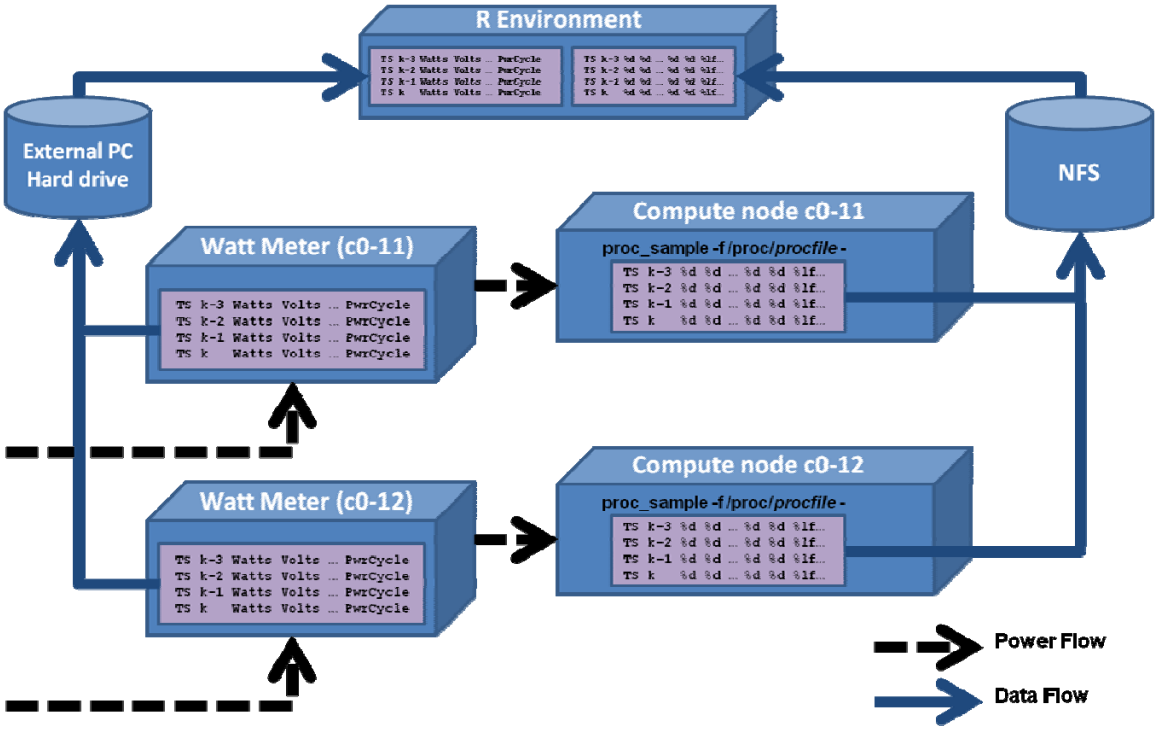


Figure 29 – Power Modeling and Analysis Environment.

Table 19 – Number of surviving variables per proc file.

/proc file	Counters	Surviving	Target
buddyinfo	33	20	16
diskstats	385	23	8
interrupts	138	20	20
loadavg	5	5	5
meminfo	30	16	5
net/dev	80	9	2
schedstat	672	416	26
slabinfo	1,216	259	31
stat	317	56	6
vmstat	51	35	11
Total	**2,927**	**859**	**130**

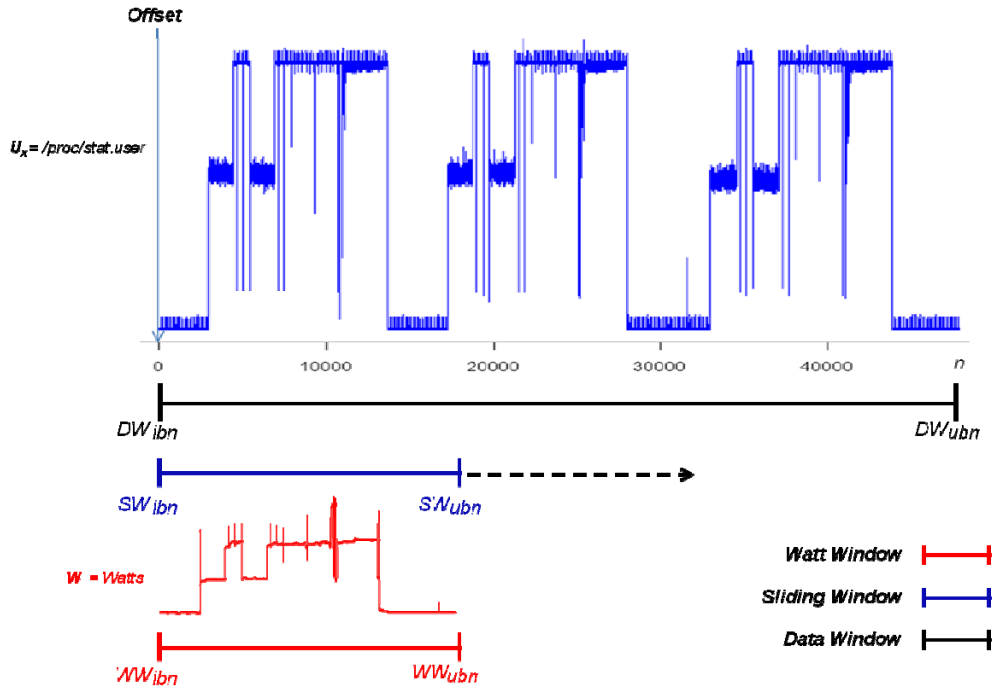


Figure 30 – Illustration of the alignment algorithm using /proc/stat.user samples.

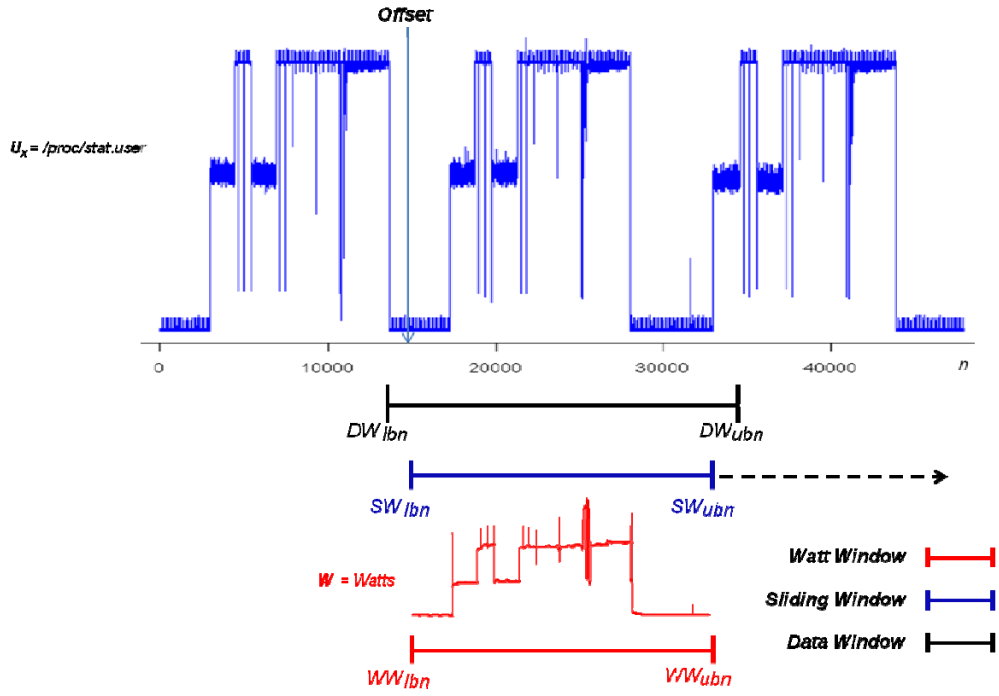


Figure 31 – Offset shows where Watt meter samples align with /proc/stat/user samples.

Table 20 – The first six columns (target variables) of **stat** in R.

user	*sys*	*sysi*	*iow*	*hirq*	*sirq*	*…*
0	0	800	0	0	0	…
0	0	799	0	0	0	…
0	0	800	0	0	0	…
0	0	800	0	0	0	…
0	0	799	0	0	0	…
39	51	689	2	1	17	…
2	4	793	0	0	0	…
43	730	27	0	0	0	…
387	413	0	0	0	0	…
430	369	0	0	0	0	…
437	364	0	0	0	0	…
427	372	0	0	0	0	…
434	366	0	0	0	0	…
433	367	0	0	0	0	…
432	368	0	0	0	0	…
…	…	…	…	…	…	…

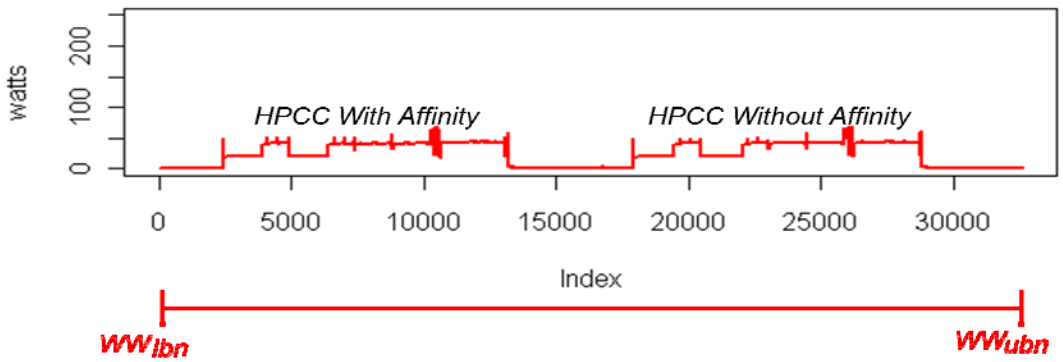


Figure 32 – Watt meter sample data for two 1-core HPCC computations.

```
model_Axi_b_window_Analysis <- function (b, b_window, A, A_window)
```

Figure 33 – R function prototype that calculates least squares regression parameters.

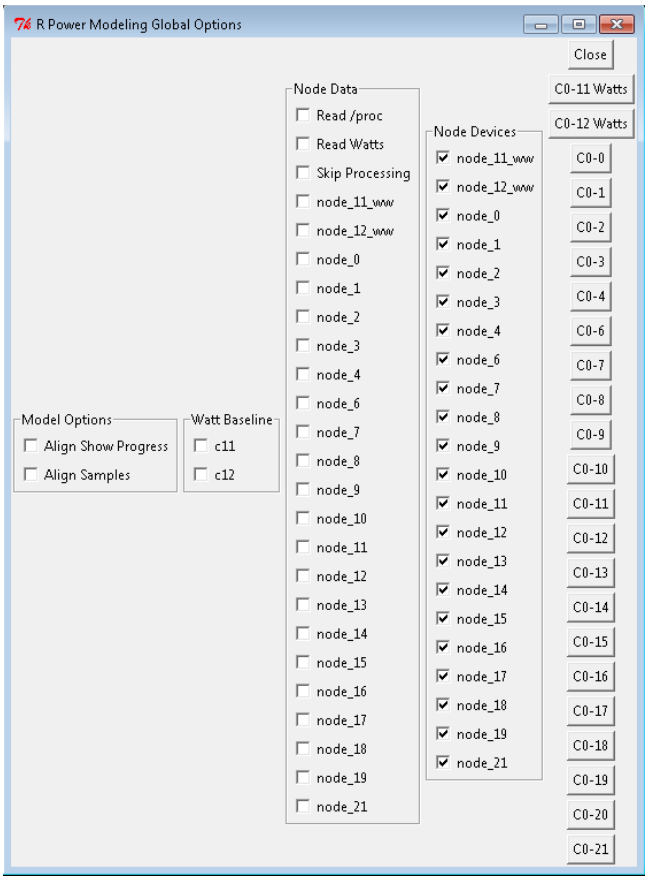


Figure 34 – R Power Modeling Tool, Main rpanel.

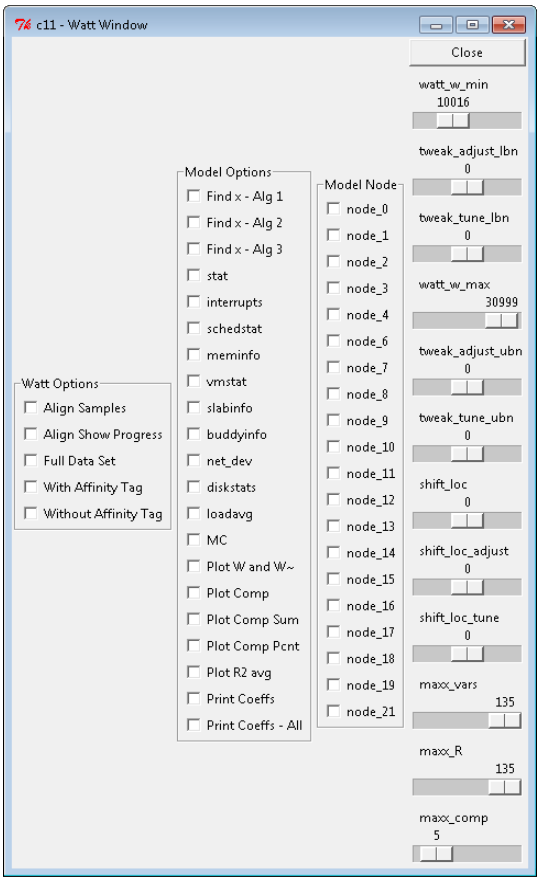


Figure 35 – RR Power Modeling Tool, Watt Window.

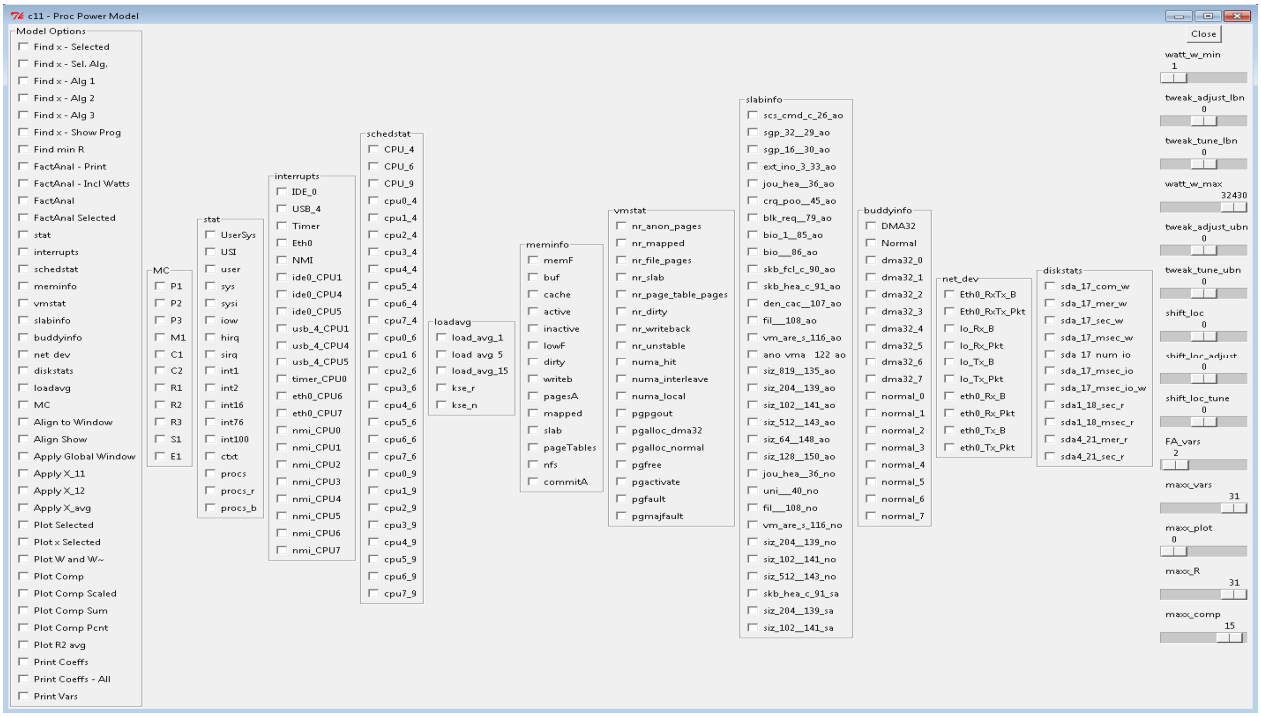


Figure 36 – R Power Modeling Tool, Compute Node Power Modeling rpanel.

FT6: Chapter 6 Tables and Figures

Table 21 – HPCC benchmark computation configurations.

Ref#	Configuration	# Compute Nodes	# Processors	# Cores	MPI Affinity	N
1	*n*1*p*1*c*1*a*1	1	1	1	1	31,000
2	*n*1*p*1*c*1*a*0	1	1	1	0	31,000
3	*n*1*p*1*c*4*a*1*s*0	1	1	4	1	31,000
4	*n*1*p*1*c*4*a*1*s*1	1	1	4	1	31,000
5	*n*1*pX*c4*a*1	1	2	4	1	31,000
6	*n*1*pX*c4*a*0*sX*	1	(1-2)	4	0	31,000
7	*n*1*p*2*c*8*a*1	1	2	8	1	31,000
8	*n*1*p*2*c*8*a*0	1	2	8	0	31,000
9	*n*2*p*4*c*16*a*1	2	4	16	1	43,000
10	*n*2*p*4*c*16*a*0	2	4	16	0	43,000
11	*n*4*p*8*c*32*a*1	4	8	32	1	61,000
12	*n*4*p*8*c*32*a*0	4	8	32	0	61,000
13	*n*20*p*40*c*160*a*1	20	40	160	1	90,000
14	*n*20*p*40*c*160*a*0	20	40	160	0	90,000

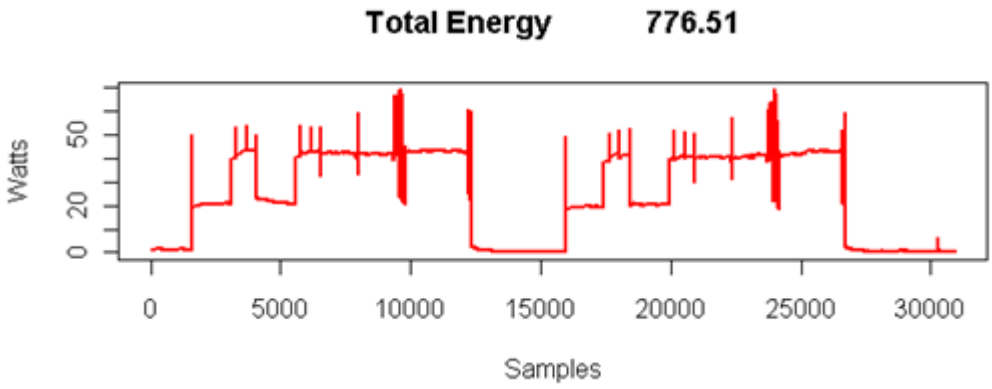


Figure 37 – Watt meter data collected from compute node c0-11.

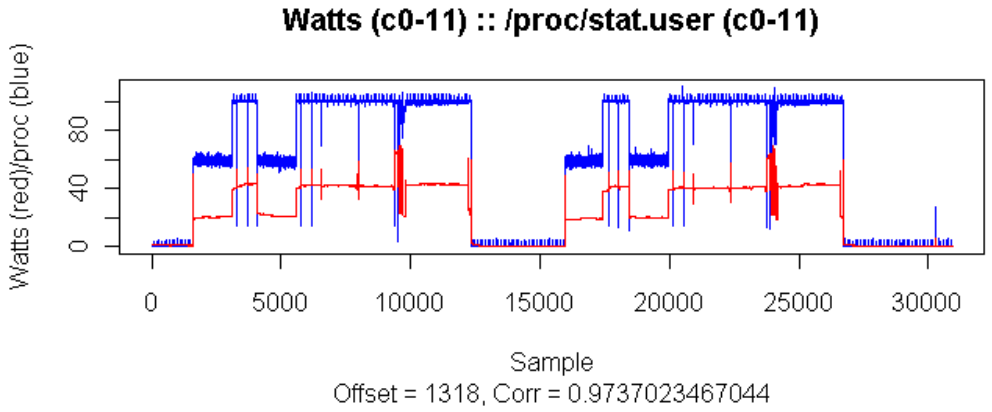


Figure 38 – Aligned Watt meter and /proc samples.

Table 22 – Target variables for *n*1*p*1*c*1*aX* configuration experiment.

/proc file	Target
buddyinfo	16
diskstats	11
interrupts	12
loadavg	-
meminfo	14
net/dev	8
schedstat	24
slabinfo	31
stat	15
vmstat	18
Total	**149**

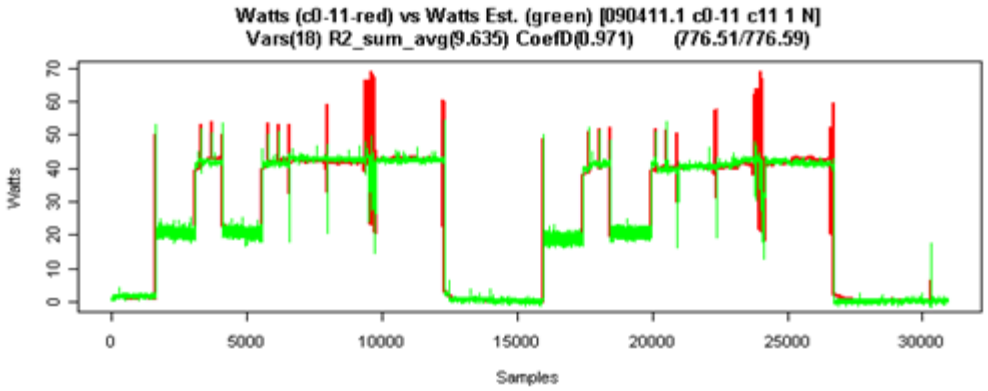
Table 23 – Least Squares results for *Algorithm 1*, *2*, and *3* (*n*1*p*1*c*1 – Contiguous HPCC).

Alg. 1 Sel. Vars	x	Est. kWs	R2_sum_avg	CoefD	R2_sum_avg_x0	Corr_X0	CoefD_X0
user	0.412276	781.93	17.517284	0.948085	17.517284	0.973702	0.948085
pgfault	0.000145	5.92	15.439988	0.954241	957.954352	0.061616	-1.839058
sys	-0.090133	-23.46	12.649698	0.962511	876.374047	-0.107748	-1.597281
active	-0.000002	-220.40	12.188212	0.963878	53.980565	0.918534	0.840020
commitA	-0.000004	-462.05	11.013654	0.967359	56.614126	0.915888	0.832215
pgfree	0.000021	0.88	10.353152	0.969317	966.190532	0.024192	-1.863467
cpu2_6	0.024169	51.03	10.215821	0.969724	434.288791	-0.121113	-0.287087
fil___108_no	0.030048	3.26	10.118123	0.970013	882.308971	0.153976	-1.614870
dma32_7	-0.005405	-132.68	10.037850	0.970251	806.989424	-0.669668	-1.391648
dma32_5	0.004680	141.90	9.976403	0.970433	806.097045	-0.668542	-1.389003
eth0_Tx_B	-0.000003	-1.21	9.926212	0.970582	961.366076	0.000982	-1.849169
int100	0.009640	1.34	9.880291	0.970718	948.834469	0.011964	-1.812029
siz_512__143_ao	-0.013255	-9.81	9.845261	0.970822	571.823899	-0.059019	-0.694696
cpu4_6	0.013017	24.42	9.821483	0.970892	382.792403	-0.123037	-0.134469
pgalloc_dma32	-0.000023	-0.21	9.806159	0.970938	964.565292	0.037126	-1.858650
nmi_CPU0	-0.721516	-18.06	9.789204	0.970988	442.444411	0.489214	-0.311258
nr_mapped	-0.005756	-96.70	9.726872	0.971173	212.278086	0.617742	0.370878
nr_anon_pages	0.000023	730.49	**9.635033**	**0.971445**	57.601797	0.914946	0.829287
Est. kWs		776.59					

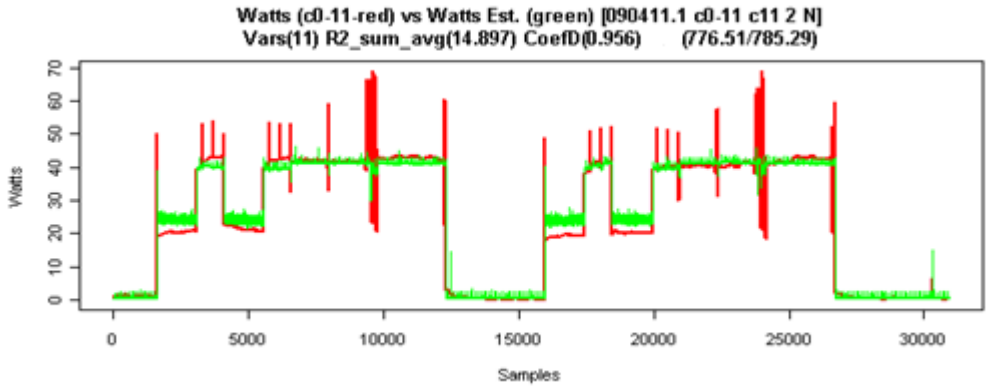
Alg. 2 Sel. Vars	x	Est. kWs	R2_sum_avg	CoefD	R2_sum_avg_x0	Corr_X0	CoefD_X0
user	0.3765905	714.24	17.517284	0.948085	17.517284	0.973702	0.948085
pgfault	0.0000764	3.13	15.439988	0.954241	957.954352	0.061616	-1.839058
pagesA	0.0000005	58.12	15.276742	0.954725	57.601510	0.914946	0.829288
pgfree	0.0000094	0.39	15.134702	0.955146	966.190532	0.024192	-1.863467
normal_3	0.0000378	5.56	15.026825	0.955465	967.667399	-0.878674	-1.867844
fil___108_no	0.0210226	2.28	14.961653	0.955659	882.308971	0.153976	-1.614870
iow	0.6478968	0.09	14.932723	0.955744	967.593784	0.006490	-1.867626
cpu3_9	0.0017003	0.58	14.906543	0.955822	948.707520	0.092729	-1.811653
cpu7_9	0.0006297	0.32	14.901006	0.955838	944.812813	0.085194	-1.800111
cpu6_9	0.0002385	0.33	14.898820	0.955845	890.794135	0.181865	-1.640017
pgalloc_normal	0.0000076	0.25	**14.896869**	**0.955851**	960.501950	0.051863	-1.846608
Est. kWs		785.29					

Alg. 3 Sel. Vars	x	Est. kWs	R2_sum_avg	CoefD	R2_sum_avg_x0	Corr_X0	CoefD_X0
user	0.3825495	725.55	17.517284	0.948085	17.517284	0.973702	0.948085
pagesA	0.0000004	46.44	17.341237	0.948606	57.601510	0.914946	0.829288
cpu5_6	0.0291761	7.53	**17.283293**	**0.948778**	499.970216	0.268409	-0.481745
Est. kWs		779.52					

Algorithm 1



Algorithm 2



Algorithm 3

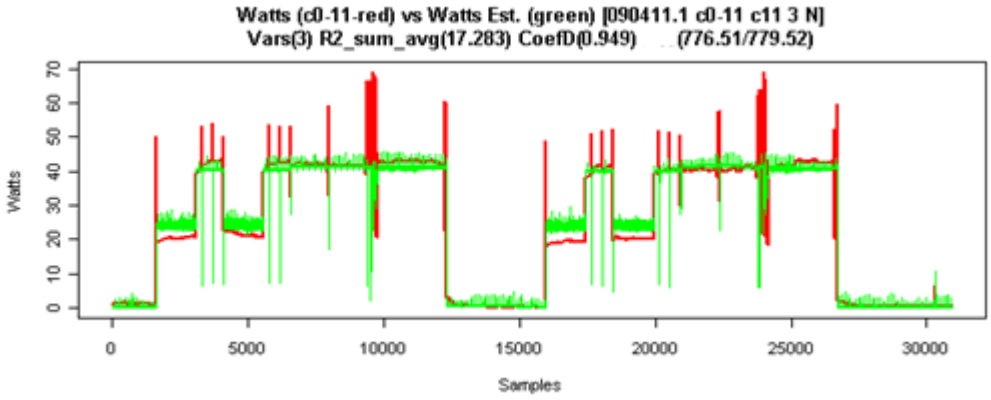
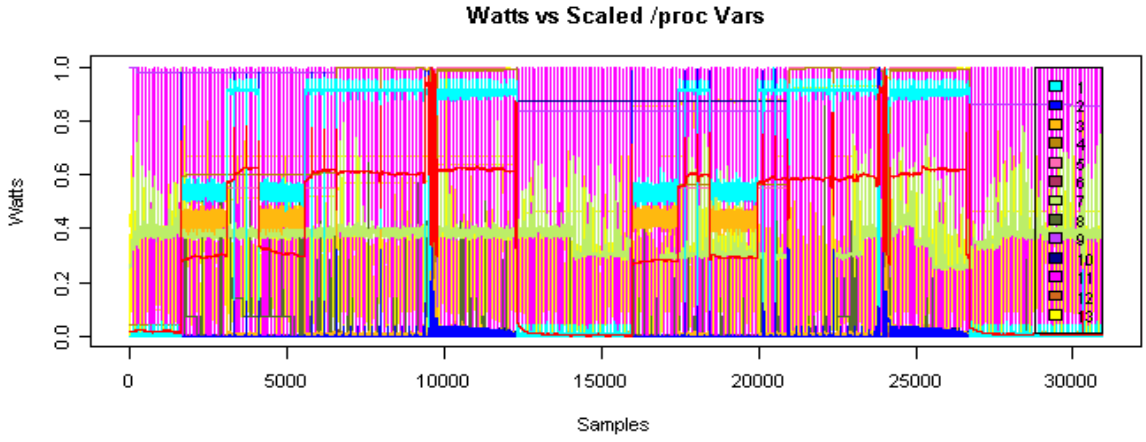
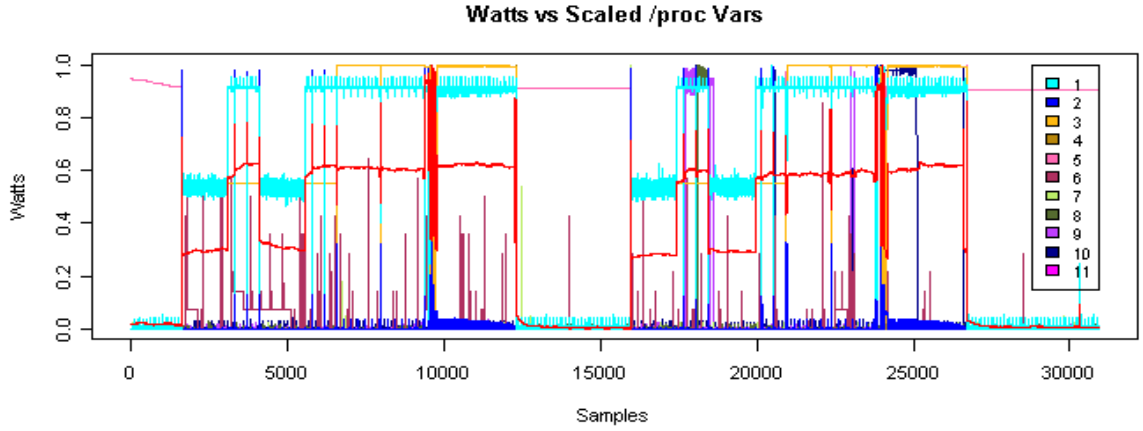


Figure 39 – Measured (red) versus estimated power dissipation (based on **Table 23**).

Algorithm 1



Algorithm 2



Algorithm 3

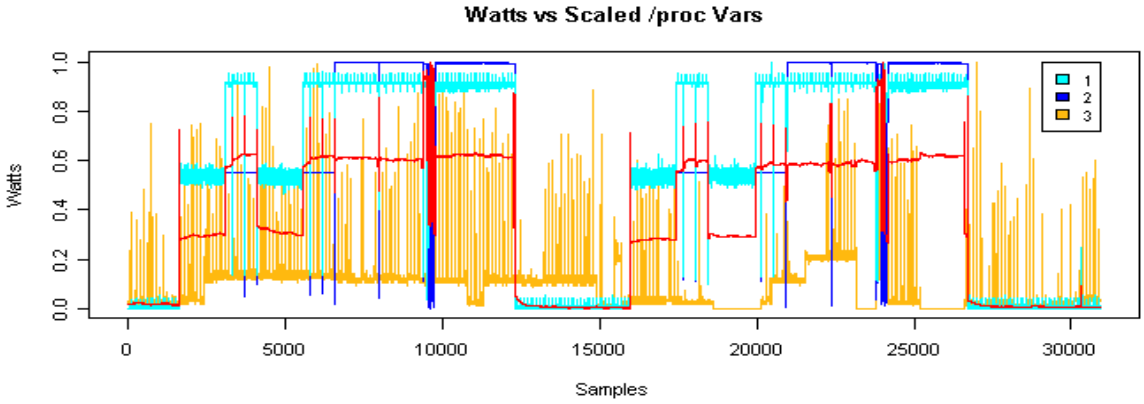
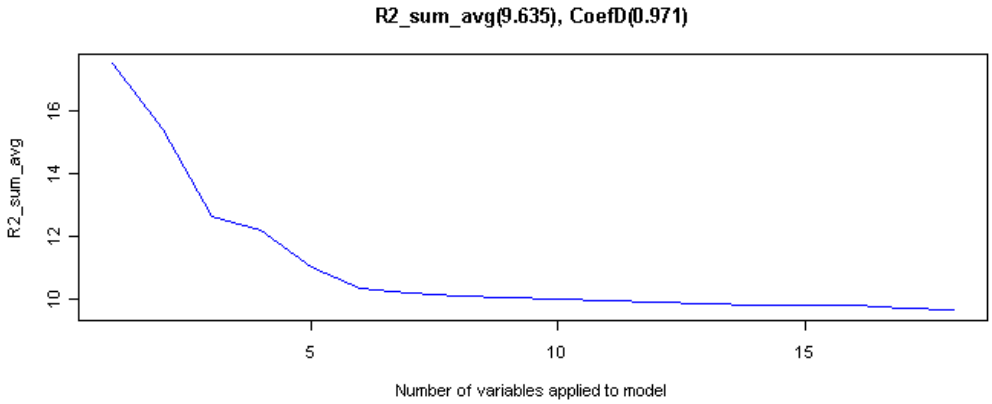
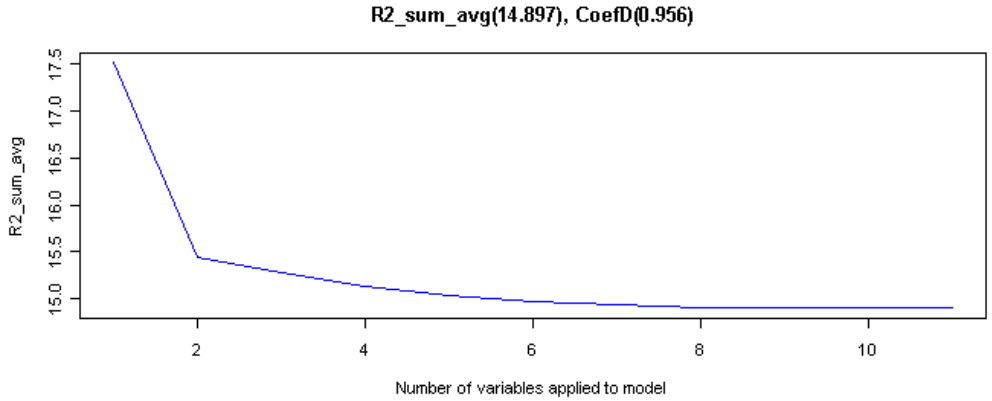


Figure 40 – Power consumption (red) versus scaled /proc variables (based on **Table 23**).

Algorithm 1



Algorithm 2



Algorithm 3

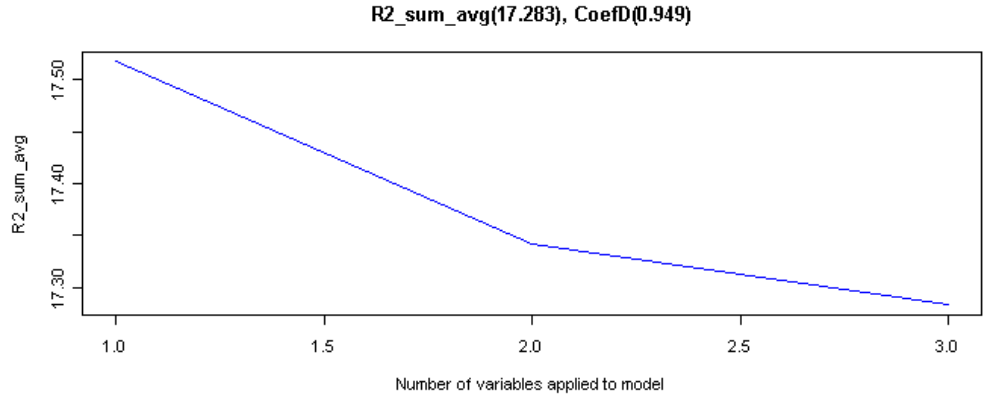


Figure 41 – Average of the sum of squared residuals versus # vars (based on **Table 23**).

Table 24 – Evaluation of power estimates ($n1c1p1aX$ - Contiguous HPCC).

	Tot. kWs	Est. kWs	% Error
Algorithm 1	776.51	776.59	(0.01)
Algorithm 2	776.51	785.29	(1.13)
Algorithm 3	776.51	779.52	(0.39)

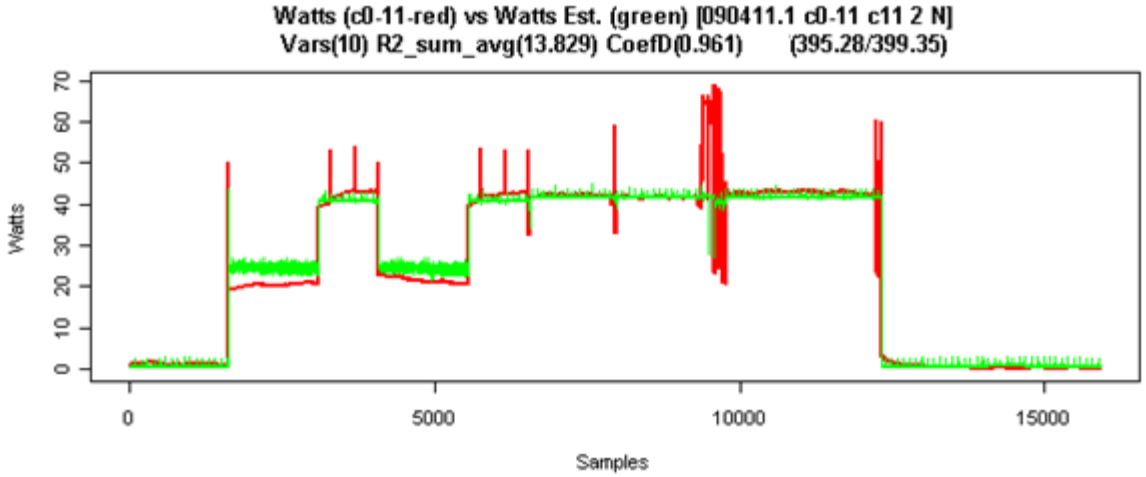


Figure 42 – Measured (red) versus estimated power load for $n1p1c1a1$.

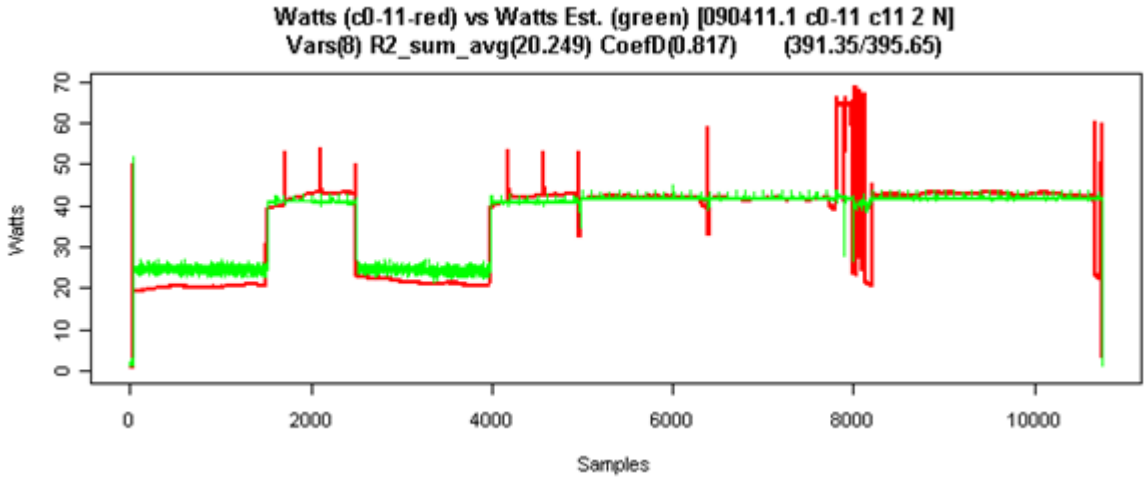


Figure 43 – Measured (red) versus estimated power load for $n1p1c1a1$ (no idle time).

Table 25 – Training variables selected by *Algorithm 2* (idle time before and after HPCC).

Alg. 2 Sel. Vars	x	Est. kWs	R2_sum_avg	CoefD	R2_sum_avg_x0	Corr_X0	CoefD_X0
user	0.39309	372.45	16.34072	0.95376	16.34072	0.97661	0.95376
pgfault	0.00008	1.72	14.20393	0.95981	959.72411	0.06502	-1.71553
cpu0_6	0.00560	1.72	14.05148	0.96024	969.80787	-0.83632	-1.74406
pagesA	0.00000	20.73	13.95344	0.96052	57.88897	0.91883	0.83620
pgfree	0.00001	0.17	13.85411	0.96080	968.58602	0.02097	-1.74060
fil___108_no	0.01009	0.78	13.83446	0.96086	848.49133	0.17728	-1.40079
procs	0.06544	0.07	13.83160	0.96086	968.29963	0.00970	-1.73979
normal_3	0.00002	1.63	13.83009	0.96087	969.78637	-0.88957	-1.74400
sgp_16__30_ao	0.04358	0.04	13.82935	0.96087	963.80317	-0.00175	-1.72707
pgalloc_normal	0.00000	0.05	**13.82900**	**0.96087**	962.31857	0.05478	-1.72287
Est. kWs		399.35					

Table 26 – Training variables selected by *Algorithm 2* (no idle time before or after HPCC).

Alg. 2 Sel. Vars	x	Est. kWs	R2_sum_avg	CoefD	R2_sum_avg_x0	Corr_X0	CoefD_X0
user	0.39324	372.24	23.79325	0.78508	23.79325	0.89598	0.78508
pgfault	0.00008	1.70	20.63050	0.81365	1420.68398	0.03580	-11.83279
pagesA	0.00000	20.59	20.48928	0.81492	85.30617	0.62003	0.22944
pgfree	0.00001	0.16	20.33920	0.81628	1433.80358	0.00287	-11.95130
cpu0_6	0.02388	0.06	20.28149	0.81680	1435.65895	-0.03086	-11.96806
fil___108_no	0.01055	0.80	20.25029	0.81708	1251.85274	-0.04460	-10.30777
sgp_16__30_ao	0.05354	0.04	20.24924	0.81709	1421.70465	0.00966	-11.84201
pgalloc_normal	0.00000	0.05	**20.24875**	**0.81710**	1424.52840	0.02860	-11.86752
Est. kWs		395.65					

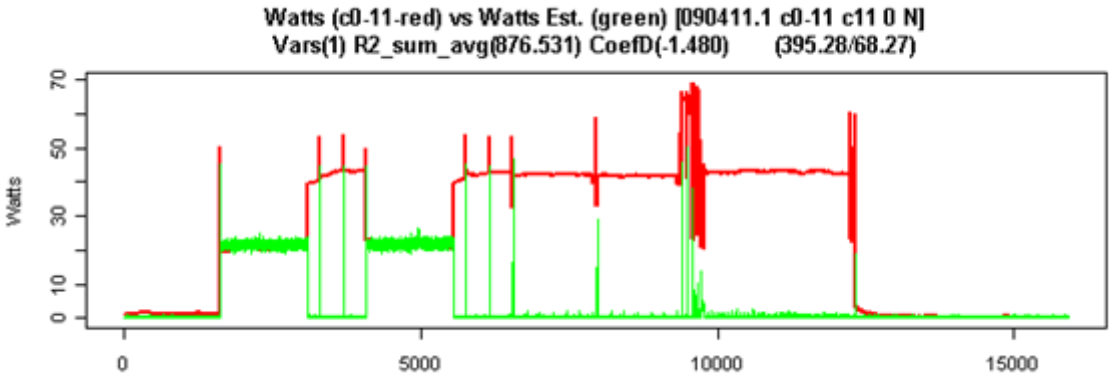


Figure 44 – /proc/stat.*sys*, *Type 2* variable.

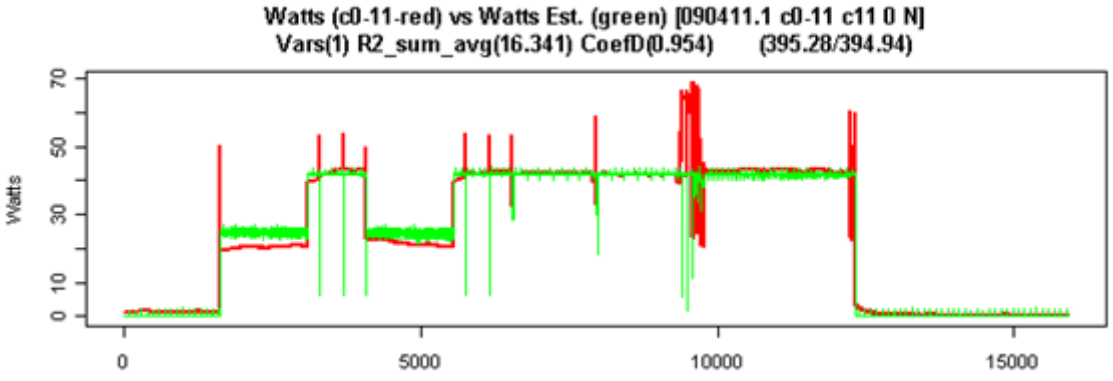


Figure 45 – /proc/stat.*user*, *Type 1* variable.

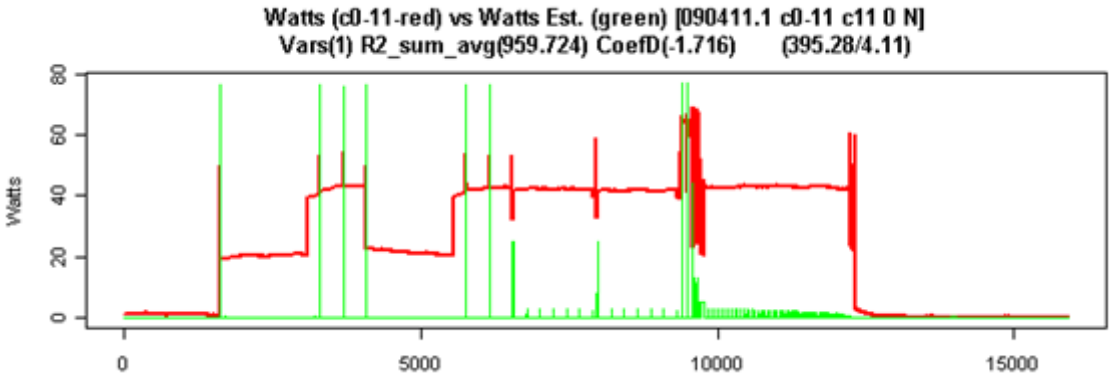


Figure 46 – /proc/vmstat.*pgfault*, *Type 2* variable.

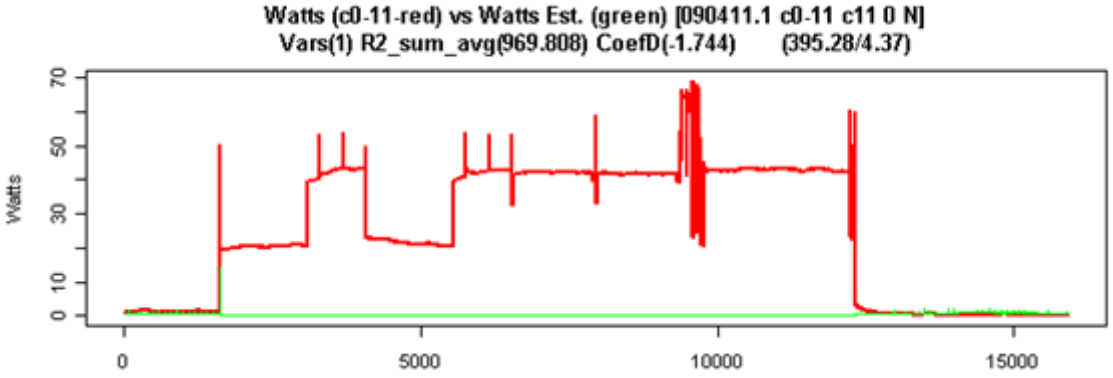


Figure 47 – /proc/schedstat.*cpu0_6*, *Type 3* variable.

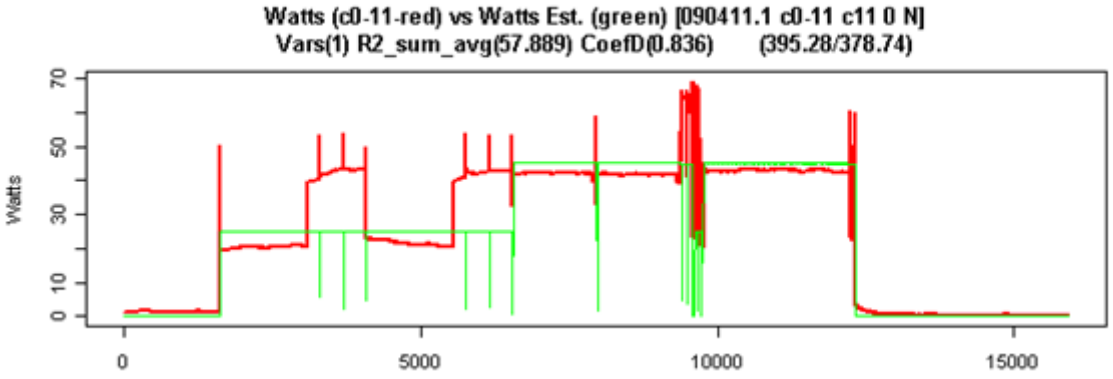


Figure 48 – /proc/meminfo.*pagesA*, *Type 1* variable.

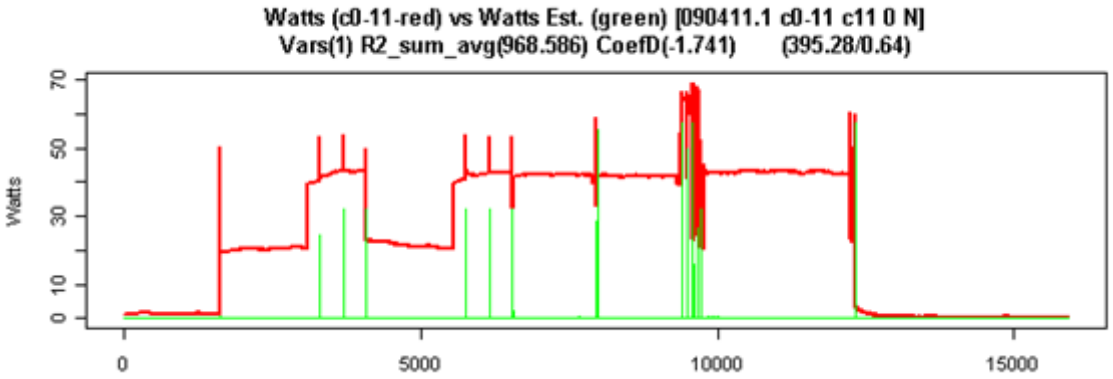


Figure 49 – /proc/vmstat.*pgfree*, *Type 2* variable.

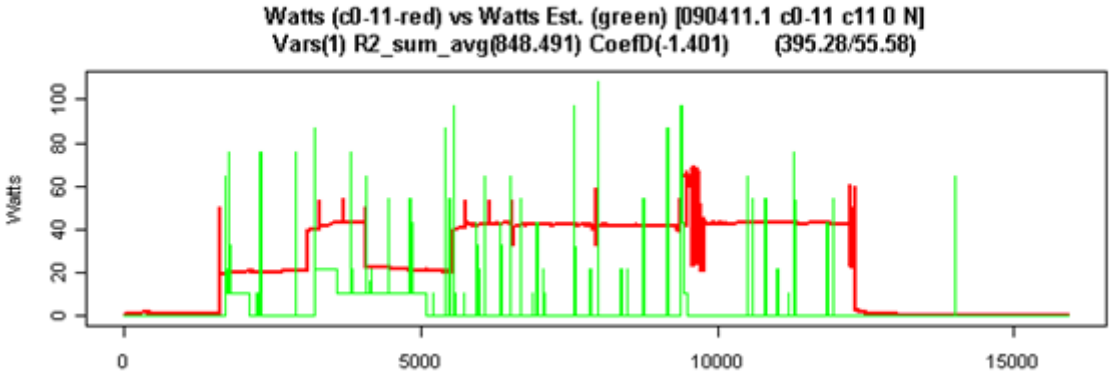


Figure 50 – /proc/slabinfo.*fil___108_no*, *Type 2* variable.

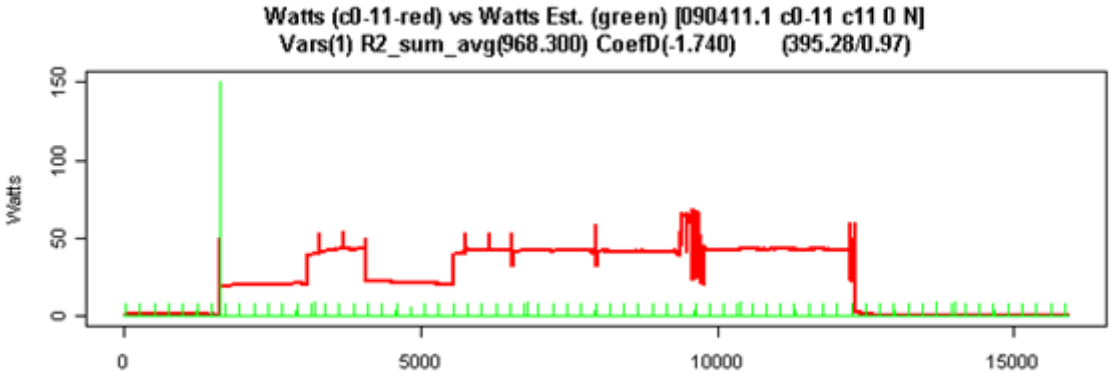


Figure 51 – /proc/stat.*procs*, *Type 3* variable.

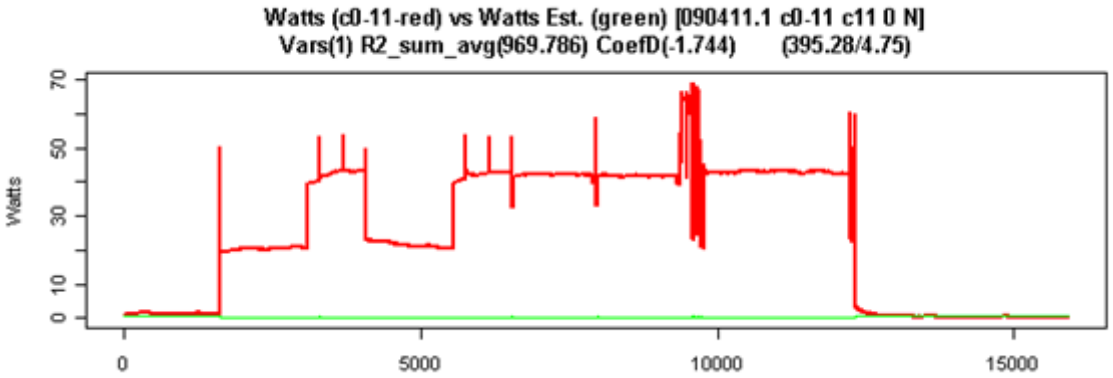


Figure 52 – /proc/buddyinfo.*normal3*, *Type 3* variable.

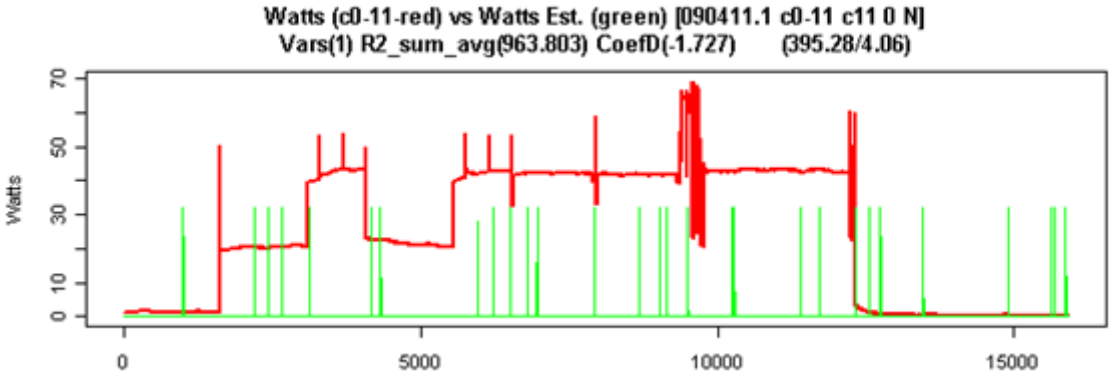


Figure 53 – /proc/slabinfo.*sgp_16__30_ao*, *Type 3* variable.

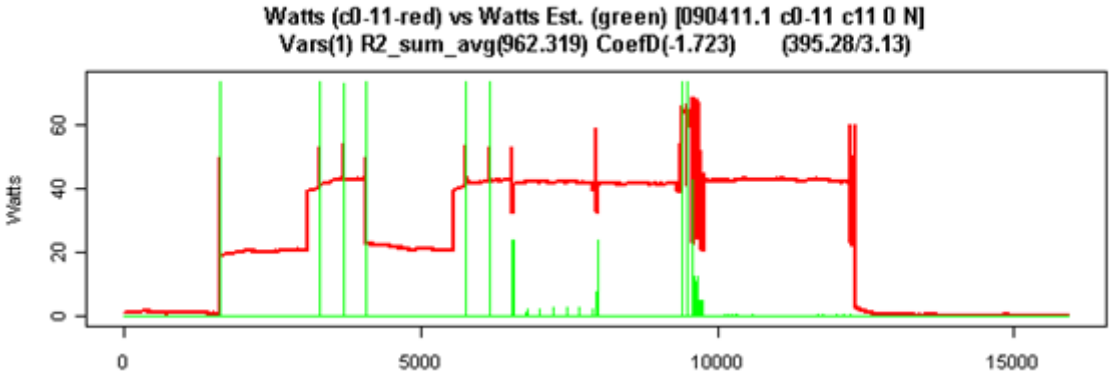


Figure 54 – /proc/vmstat.*pgalloc_normal*, *Type 2* variable.

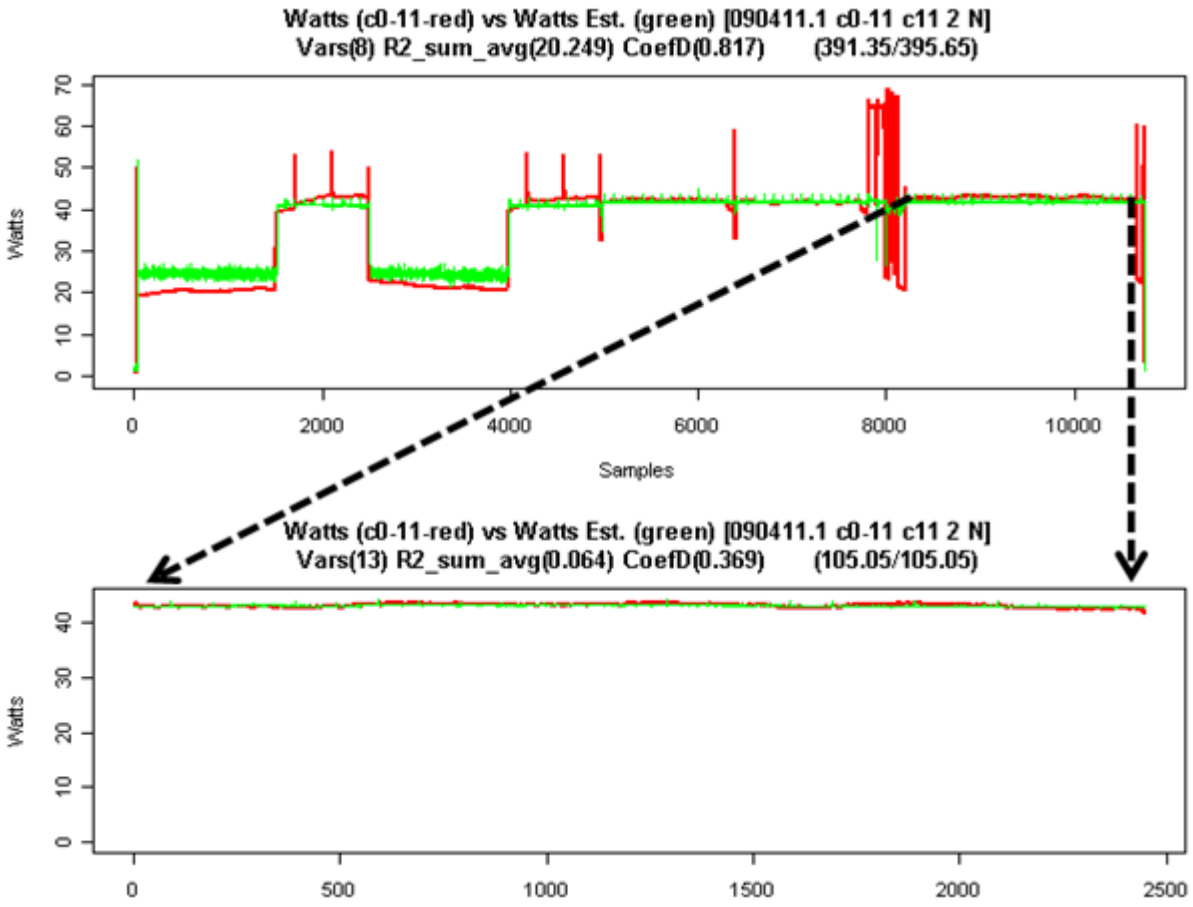


Figure 55 – Measured (red) versus estimated power load (*n*1*p*1*c*1*a*1 HPCC HPC phase).

Table 27 – *Algorithm 2* selected variables for the *n*1*p*1*c*1*a*1 computation shown in **Figure 55** – .

Alg. 2 Sel. Vars	x	Est. kWs	R2_sum_avg	CoefD	R2_sum_avg_x0	Corr_X0	CoefD_X0
commitA	0.000005	92.3450	0.07953	0.21061	0.07953	0.49309	0.21061
cpu7_6	0.015383	0.3669	0.07301	0.27529	708.24031	0.36796	-7028.78440
normal_1	0.01089	0.4917	0.06760	0.32903	293.55704	-0.04667	-2912.76062
jou_hea__36_no	0.001864	0.1498	0.06567	0.34819	734.13659	0.15648	-7285.82325
sysi	0.056723	8.8110	0.06486	0.35624	0.40412	-0.01692	-3.01114
cpu3_9	0.016518	0.0177	0.06438	0.36095	1664.70799	0.14380	-16522.40053
fil___108_no	0.001555	0.0092	0.06405	0.36422	1770.83810	0.07246	-17575.81669
inactive	0.000026	2.8252	0.06380	0.36678	1.07717	-0.05731	-9.69166
sda_17_msec_io	0.009475	0.0049	0.06360	0.36875	1804.45821	0.03152	-17909.52007
siz_512__143_ao	0.000375	0.0228	0.06356	0.36908	549.98108	0.11270	-5457.95005
lo_Rx_B	0.000017	0.0019	0.06356	0.36912	1570.23027	0.01740	-15584.64262
IDE_0	0.000161	0.0035	0.06356	0.36914	921.74737	0.00026	-9147.99255
sda_17_msec_w	0.000151	0.0001	**0.06356**	**0.36914**	1774.14012	0.03525	-17608.59161
Est. kWs		105.0496					

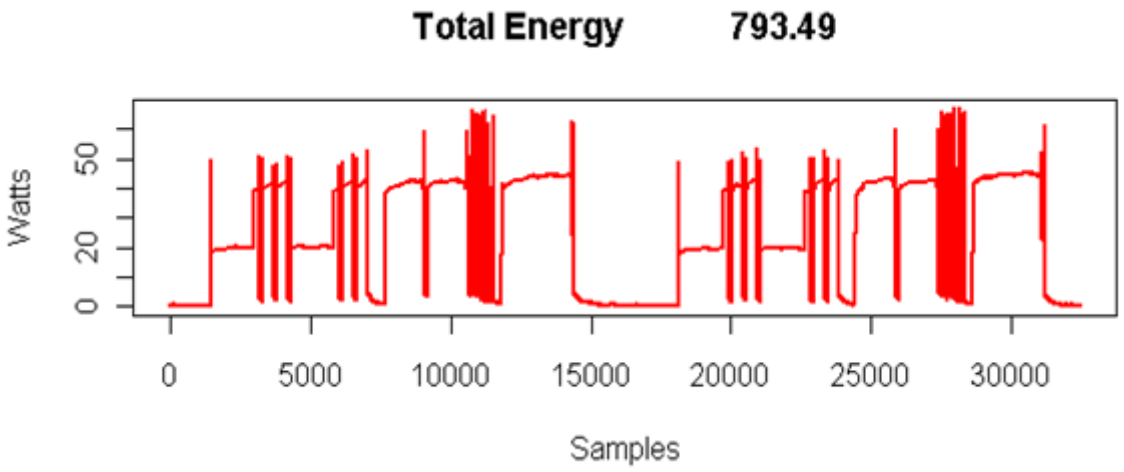


Figure 56 – Watt meter power load samples ($n1p1c1aX$, Split HPCC).

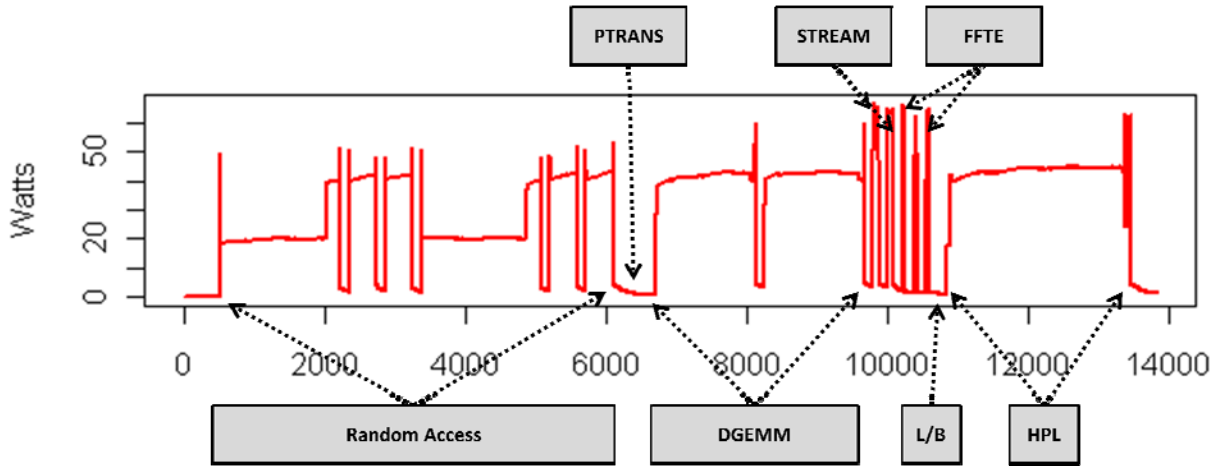


Figure 57 – Watt meter power load samples ($n1p1c1a1$, Split HPCC, left part of **Figure 56**).

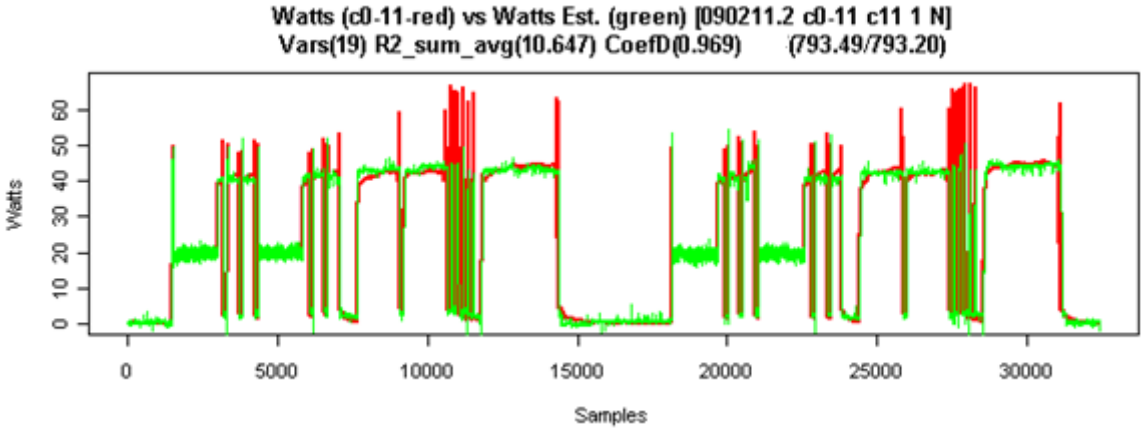
Table 28 – Least Squares results for *Algorithm 1*, *2*, and *3* (Split HPCC benchmarks, *n*1*c*1*p*1*aX*).

Alg. 1 Sel. Vars	x	Est. kWs	R2_sum_avg	CoefD	R2_sum_avg_x0	Corr_X0	CoefD_X0
user	0.349191	663.51	20.265715	0.941730	20.265715	0.970548	0.941730
sys	-0.146360	-39.68	17.419225	0.949915	863.996839	-0.110260	-1.484231
numa_hit	0.000144	5.99	15.215736	0.956251	938.531626	0.047598	-1.698539
pageTables	0.000101	26.57	13.913116	0.959996	51.891448	0.924410	0.850798
commitA	-0.000004	-506.92	11.912514	0.965748	55.548941	0.920777	0.840281
nr_mapped	0.005434	74.95	11.565939	0.966745	198.364697	0.679351	0.429646
pgfree	0.000014	0.59	11.329935	0.967423	945.763307	0.005345	-1.719332
cpu3_9	-0.005311	-0.65	11.213948	0.967757	941.552143	0.036466	-1.707224
nmi_CPU7	0.838958	4.32	11.107985	0.968061	792.850318	0.305405	-1.279665
cpu1_9	0.003070	0.87	11.023809	0.968303	929.456271	0.097567	-1.672445
cpu3_6	0.048102	18.68	10.946229	0.968527	664.241929	-0.113758	-0.909880
ctxt	-0.001466	-2.59	10.868700	0.968749	773.159978	-0.002008	-1.223050
pagesA	0.000004	557.98	10.813443	0.968908	54.425526	0.922993	0.843512
pgalloc_dma32	-0.000036	-0.34	10.774938	0.969019	943.148708	0.038265	-1.711814
jou_hea__36_no	-0.002483	-2.74	10.752593	0.969083	632.336263	-0.049808	-0.818142
cpu7_6	0.033729	11.95	10.731150	0.969145	720.444453	-0.178844	-1.071478
siz_204__139_no	-0.037011	-11.16	10.700790	0.969232	514.484232	-0.024009	-0.479285
pgactivate	-0.013452	-1.21	10.671886	0.969315	940.211072	-0.000105	-1.703368
normal_3	-0.000051	-6.93	**10.646725**	**0.969388**	945.798118	-0.862265	-1.719432
Est. kWs		793.20					

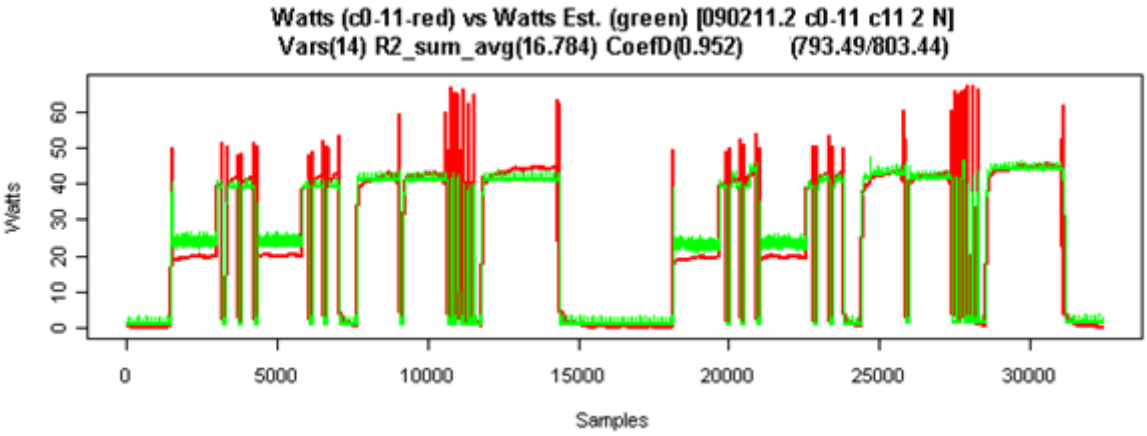
Alg. 2 Sel. Vars	x	Est. kWs	R2_sum_avg	CoefD	R2_sum_avg_x0	Corr_X0	CoefD_X0
user	0.364451	692.50	20.265715	0.941730	20.265715	0.970548	0.941730
numa_hit	0.000071	2.95	18.949733	0.945514	938.531626	0.047598	-1.698539
inactive	0.000002	3.69	18.040396	0.948129	661.063576	0.214652	-0.900741
active	0.000001	79.38	17.753262	0.948954	52.187176	0.923554	0.849947
normal_2	**0.000101**	**12.23**	**17.509286**	**0.949656**	**945.835722**	**-0.867266**	**-1.719540**
cpu7_9	0.002892	7.30	17.220951	0.950485	794.528645	0.306561	-1.284490
cpu1_9	0.004692	1.33	17.060254	0.950947	929.456271	0.097567	-1.672445
pgfree	0.000010	0.43	16.912715	0.951371	945.763307	0.005345	-1.719332
cpu5_9	0.001654	3.21	16.787346	0.951732	839.227540	0.241185	-1.413012
eth0_Rx_B	0.000004	0.15	16.785599	0.951737	938.414463	0.011079	-1.698202
fil___108_no	0.003688	0.18	16.784018	0.951741	924.352487	0.069726	-1.657770
normal_0	**0.000002**	**0.06**	**16.783951**	**0.951741**	**946.018430**	**-0.199433**	**-1.720066**
pgalloc_normal	0.000001	0.02	16.783935	0.951742	941.060526	0.036460	-1.705810
cpu1_4	0.000351	0.00	**16.783923**	**0.951742**	946.118612	0.012043	-1.720354
Est. kWs		803.44					

Alg. 3 Sel. Vars	x	Est. kWs	R2_sum_avg	CoefD	R2_sum_avg_x0	Corr_X0	CoefD_X0
user	0.363445	690.59	20.265715	0.941730	20.265715	0.970548	0.941730
cpu7_9	0.002003	5.06	19.580602	0.943700	794.528645	0.306561	-1.284490
pageTables	0.000279	73.06	19.132207	0.944990	51.891448	0.924410	0.850798
mapped	0.000301	16.59	19.027420	0.945291	198.364697	0.679351	0.429646
procs_r	0.076919	7.19	19.009031	0.945344	368.101447	0.323968	-0.058394
nmi_CPU7	0.413526	2.13	**19.008336**	**0.945346**	792.850318	0.305405	-1.279665
Est. kWs		794.62					

Algorithm 1



Algorithm 2



Algorithm 3

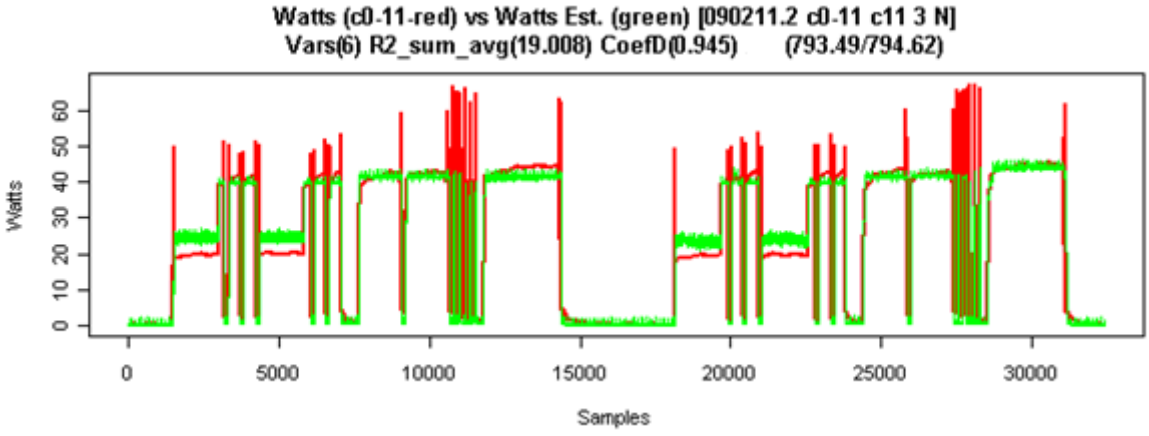
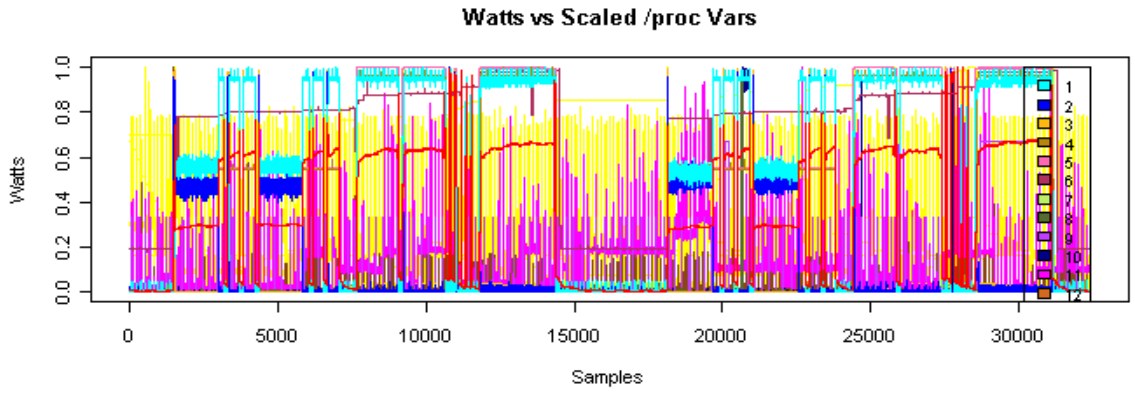
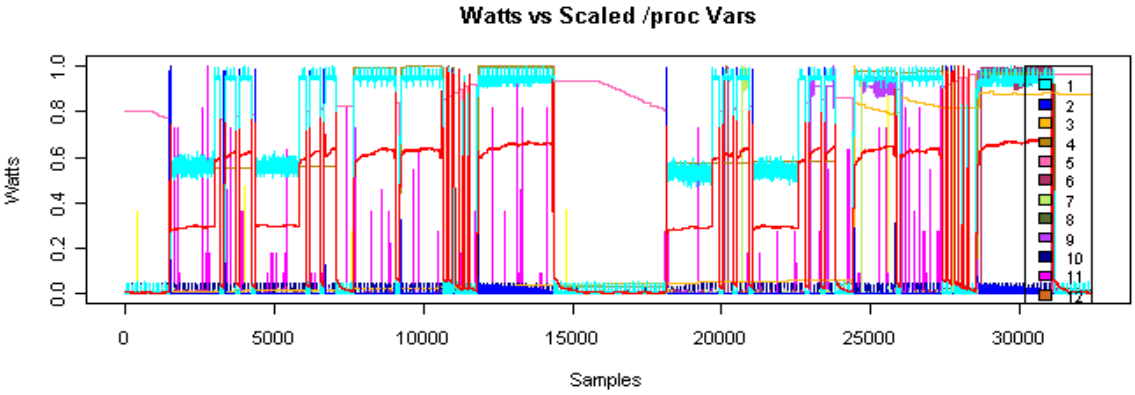


Figure 58 – Measured (red) versus estimated power load (based on **Table 28**).

Algorithm 1



Algorithm 2



Algorithm 3

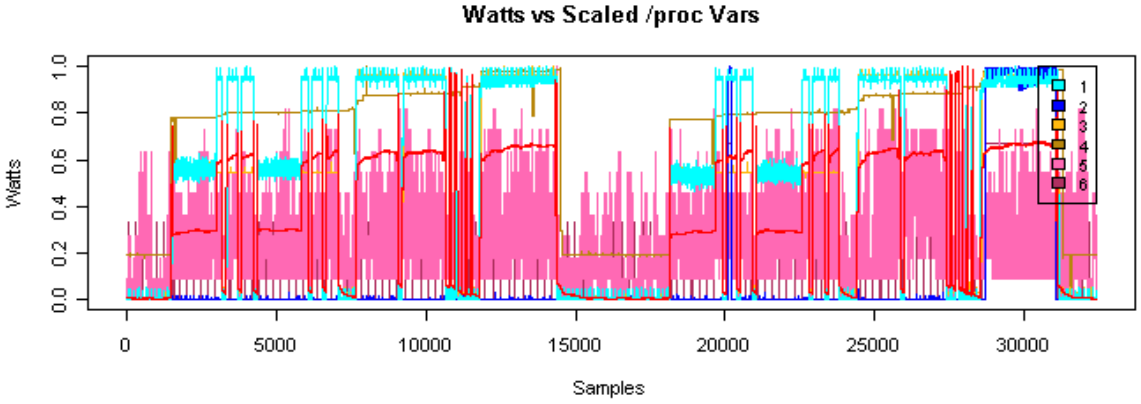
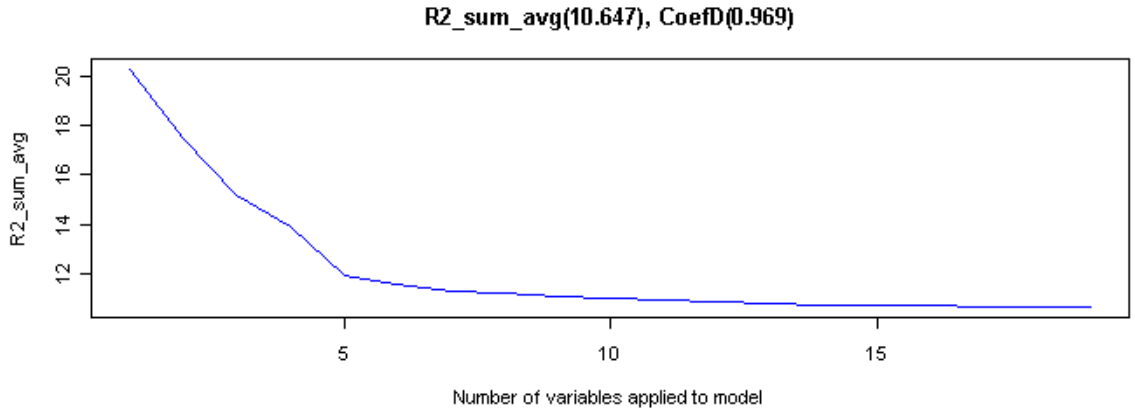
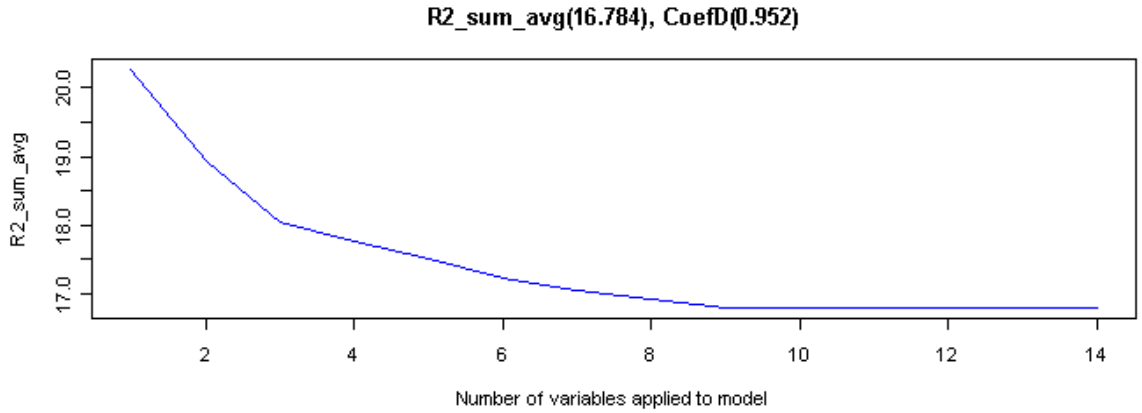


Figure 59 – Power consumption (red) versus scaled /proc variables (based on **Table 28**).

Algorithm 1



Algorithm 2



Algorithm 3

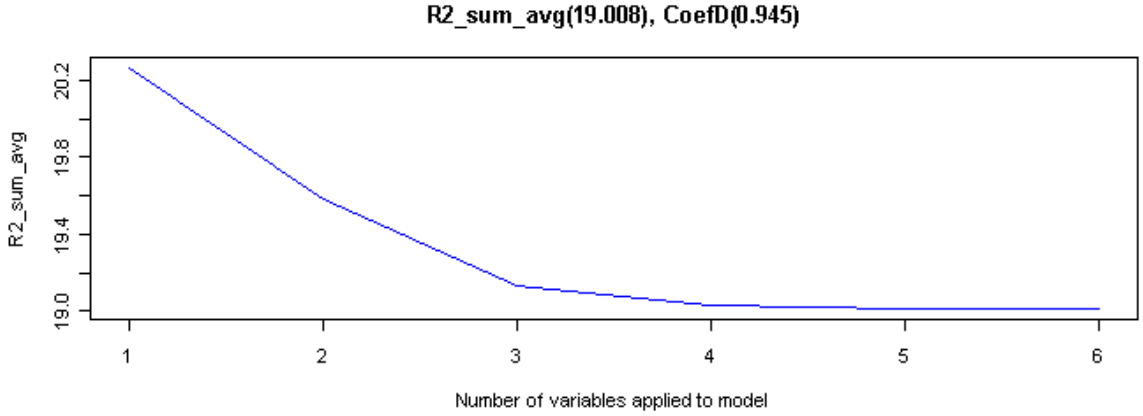


Figure 60 – Average of the sum of squared residuals versus # vars (based on **Table 28**).

Table 29 – Evaluation of power estimates obtained for split HPCC benchmarks, $n1c1p1aX$.

	Tot. kWs	Est. kWs	% Error
Algorithm 1	793.49	793.20	0.04
Algorithm 2	793.49	803.44	(1.25)
Algorithm 3	793.49	794.62	(0.14)

Table 30 – Least Squares results for Algorithm 3 ($n1c1p1$, Split HPCC, CORR = 0.001).

Alg. 3 Sel. Vars	x	Est. kWs	R2_sum_avg	CoefD	R2_sum_avg_x0	Corr_X0	CoefD_X0
user	0.360959	685.87	20.265720	0.941730	20.265720	0.970550	0.941730
numa_hit	0.000070	2.90	18.949730	0.945510	938.531630	0.047600	-1.698540
inactive	0.000004	9.38	18.040400	0.948130	661.063580	0.214650	-0.900740
active	0.000001	76.40	17.753260	0.948950	52.187180	0.923550	0.849950
nmi_CPU7	1.194600	6.15	17.563990	0.949500	792.850320	0.305410	-1.279660
pgfree	0.000011	0.45	17.390150	0.950000	945.763310	0.005350	-1.719330
cpu1_9	0.004720	1.34	17.227170	0.950470	929.456270	0.097570	-1.672440
cpu5_9	0.001383	2.69	17.148730	0.950690	839.227540	0.241190	-1.413010
mapped	0.000151	8.33	17.131730	0.950740	198.364700	0.679350	0.429650
eth0_Rx_B	0.000007	0.26	17.126640	0.950760	938.414460	0.011080	-1.698200
fil___108_no	0.004083	0.20	17.124650	0.950760	924.352490	0.069730	-1.657770
procs_r	0.010811	1.01	17.124310	0.950760	368.101450	0.323970	-0.058390
cpu1_4	0.000756	0.01	17.124250	0.950760	946.118610	0.012040	-1.720350
pgalloc_normal	0.000001	0.02	17.124240	0.950760	941.060530	0.036460	-1.705810
siz_102__141_no	0.000140	0.03	17.124240	0.950760	788.505520	0.012120	-1.267170
Est. kWs		795.04					

Table 31 – Factor loadings of variables tracked by /proc/buddyinfo ($n1p1c1aX$, Split HPCC).

Variable	Factor1	Factor2	Factor3	Factor4	Factor5	Factor6
dma32_3	**0.96**	0.28	0.05	0.05	0.00	0.00
dma32_4	**0.96**	0.27	0.03	0.03	-0.03	-0.01
dma32_5	**0.96**	0.27	0.02	0.01	-0.04	-0.02
dma32_6	**0.96**	0.27	0.03	0.03	-0.04	-0.01
dma32_2	**0.95**	0.29	0.05	0.06	0.02	0.00
dma32_7	**0.95**	0.28	0.04	0.06	-0.04	0.00
dma32_1	**0.94**	0.30	0.06	0.07	0.10	0.02
dma32_0	**0.73**	0.22	**0.52**	-0.14	0.17	0.01
normal_2	0.26	**0.95**	0.10	0.13	0.01	-0.06
normal_5	0.27	**0.95**	0.10	0.12	0.00	0.06
normal_3	0.26	**0.94**	0.12	0.17	0.03	-0.02
normal_4	0.26	**0.94**	0.12	0.17	0.03	0.02
normal_6	0.27	**0.94**	0.10	0.12	0.00	0.10
normal_7	0.38	**0.78**	-0.05	-0.28	-0.04	-0.12
normal_1	0.12	**0.45**	**0.35**	0.72	0.00	0.00
normal_0	0.03	**0.14**	**0.97**	0.20	-0.02	0.00
Watts	**-0.54**	**-0.75**	**-0.07**	-0.05	0.05	0.06

Table 32 – Factor loadings of variables tracked by /proc/diskstats ($n1p1c1aX$, split HPCC).

Variable	Factor1	Factor2
sda_17_sec_w	**0.99**	0.09
sda1_18_sec_r	**0.93**	0.11
sda_17_mer_w	**0.91**	-0.05
sda_17_com_w	**0.82**	0.23
sda4_21_sec_r	**0.67**	-0.01
sda_17_msec_w	0.29	**0.92**
sda_17_msec_io_w	0.27	**0.96**
sda_17_msec_io	0.21	**0.86**
sda_17_num_io	0.00	0.01
Watts	0.00	0.01

Table 33 – Factor loadings of variables tracked by /proc/interrupts ($n1p1c1aX$, split HPCC).

Variable	Factor1	Factor2	Factor3	Factor4	Factor5	Factor6
nmi_CPU0	**0.87**	-0.24	-0.23	-0.28	-0.11	-0.22
Watts	**0.80**	0.39	0.29	0.19	0.16	0.25
nmi_CPU7	-0.04	**0.99**	-0.07	-0.05	-0.04	-0.05
nmi_CPU5	-0.01	-0.02	**0.99**	-0.07	-0.07	-0.05
nmi_CPU2	-0.02	-0.05	-0.02	**1.00**	0.00	-0.02
nmi_CPU4	-0.04	-0.01	0.02	-0.05	**0.99**	-0.09
nmi_CPU1	0.01	-0.01	0.00	-0.02	0.02	**0.36**
nmi_CPU6	-0.01	0.00	0.00	-0.02	0.01	**0.23**
nmi_CPU3	0.00	-0.01	-0.01	-0.01	0.01	**0.17**
usb_4_CPU4	0.00	-0.01	0.00	0.01	0.00	0.00
ide0_CPU4	0.00	0.00	0.00	0.00	0.00	0.00
timer_CPU0	0.00	0.00	0.00	0.00	0.00	0.00
eth0_CPU6	0.00	0.00	0.00	0.00	-0.01	0.00

Table 34 – Factor loadings of variables tracked by /proc/loadavg ($n1p1c1aX$, split HPCC).

Variable	Factor1	Factor2
load_avg_1	**0.92**	0.38
load_avg_5	**0.70**	**0.68**
Watts	**0.69**	0.37
load_avg_15	0.38	**0.92**
kse_n	0.51	**0.51**
kse_r	0.21	0.13

Table 35 – Factor loadings for /proc/meminfo ($n1p1c1aX$, Split HPCC).

Variable	Factor1	Factor2	Factor3	Factor4	Factor5
active	**0.99**	0.05	0.10	0.08	-0.06
pagesA	**0.99**	0.04	0.13	0.08	-0.05
pageTables	**0.98**	0.04	0.12	0.10	-0.05
commitA	**0.98**	0.04	0.13	0.10	-0.04
Watts	**0.91**	0.06	0.07	0.13	-0.02
mapped	**0.63**	0.16	-0.05	**0.75**	-0.03
cache	-0.01	**0.96**	0.17	-0.09	0.17
slab	0.10	**0.90**	0.26	0.05	0.11
buf	-0.08	**0.69**	-0.53	-0.22	-0.40
inactive	0.14	**0.27**	**0.88**	0.12	0.32
dirty	0.01	0.05	0.00	0.02	-0.01
memF	**-0.98**	-0.06	-0.13	-0.08	0.04
nfs	-0.01	0.00	0.00	0.00	0.01
writeb	0.00	0.00	0.02	0.00	0.00

Table 36 – Factor loadings of net/dev and stat ($n1p1c1aX$, Split HPCC).

Variable	Factor1	Factor2	Factor3
int100	**0.98**	0.12	0.11
eth0_CPU6	**0.98**	0.12	0.11
int1	**0.94**	0.12	0.11
eth0_Tx_Pkt	0.10	**0.99**	0.04
eth0_Rx_Pkt	0.12	**0.98**	-0.11
lo_Tx_Pkt	0.09	0.00	0.03
Watts	0.00	0.00	-0.01

Table 37 – Factor loadings of schedstat ($n1p1c1aX$, split HPCC).

Variable	Factor1	Factor2	Factor3	Factor4	Factor5	Factor6
cpu0_4	**0.90**	0.01	-0.14	-0.24	-0.33	-0.06
cpu0_9	**0.90**	0.01	-0.14	-0.24	-0.33	-0.06
Watts	**0.62**	-0.02	0.11	0.13	0.30	-0.04
cpu3_4	0.03	**0.98**	0.00	0.06	0.03	0.18
cpu1_4	0.05	**0.82**	0.00	0.07	0.03	0.57
cpu5_4	-0.09	**0.75**	-0.02	0.00	-0.01	-0.30
cpu6_4	-0.01	**0.50**	0.00	0.02	0.01	-0.07
cpu7_4	0.08	**0.28**	0.00	-0.06	0.53	0.49
cpu4_9	0.00	0.01	**1.00**	-0.02	-0.04	-0.01
cpu4_4	0.00	0.00	**1.00**	-0.02	-0.04	0.00
cpu2_9	-0.03	-0.06	-0.03	**0.99**	0.08	-0.05
cpu2_4	0.00	0.17	-0.01	**0.37**	0.04	0.01
cpu0_6	**-0.85**	-0.01	0.14	**0.20**	0.40	0.05
cpu5_6	0.10	0.00	-0.04	**0.20**	-0.21	-0.01
cpu3_6	-0.11	-0.01	0.04	**0.17**	-0.05	0.01
cpu7_9	0.06	-0.02	-0.01	-0.18	**0.98**	-0.02
cpu1_9	-0.08	-0.03	-0.01	-0.03	-0.02	**0.21**
cpu2_6	-0.07	0.05	0.09	**-0.91**	0.08	0.05
cpu5_9	-0.27	0.02	-0.04	-0.09	-0.07	0.02
cpu4_6	0.06	0.03	-0.68	0.02	-0.11	0.01
cpu7_6	0.05	0.02	-0.03	-0.08	-0.32	0.01
cpu6_6	0.04	0.01	-0.01	-0.01	0.05	0.00
cpu6_9	-0.08	0.00	-0.01	-0.03	-0.02	0.00
cpu1_6	0.24	0.02	-0.05	-0.15	-0.03	-0.03
cpu3_9	-0.07	0.05	-0.01	-0.02	-0.02	-0.05

Table 38 – Factor loadings of variables tracked by /proc/slabinfo (*n*1*p*1*c*1*aX*, split HPCC).

Variable	Factor1	Factor2	Factor3	Factor4	Factor5	Factor6
siz_819__135_ao	**0.98**	0.01	0.01	-0.12	0.04	-0.03
vm_are_s_116_no	**0.93**	0.02	0.02	-0.16	0.05	0.33
ano_vma__122_ao	**0.92**	0.01	0.01	0.34	0.05	0.00
vm_are_s_116_ao	**0.91**	0.04	0.13	0.34	0.09	0.00
siz_128__150_ao	**0.74**	0.04	0.49	0.01	0.14	-0.06
den_cac__107_ao	**0.70**	0.05	-0.01	0.20	0.52	-0.03
Watts	**0.63**	-0.01	-0.02	-0.10	0.01	-0.13
siz_204__139_ao	**0.46**	0.00	0.27	0.54	0.24	0.01
bio___86_ao	0.01	**0.89**	0.01	0.10	0.01	0.03
bio_1__85_ao	0.01	**0.83**	0.01	0.10	0.02	0.00
blk_req__79_ao	-0.02	**0.79**	0.01	0.02	0.07	-0.01
crq_poo__45_ao	0.00	**0.74**	0.00	-0.01	0.05	-0.04
scs_cmd_c_26_ao	0.02	**0.73**	0.01	0.01	0.06	0.02
jou_hea__36_ao	0.05	**0.49**	0.01	0.54	0.11	0.06
skb_fcl_c_90_ao	-0.01	0.00	**0.91**	-0.01	0.03	-0.03
siz_64__148_ao	0.14	0.03	**0.82**	0.19	0.09	-0.01
siz_102__141_ao	0.08	-0.01	**0.77**	0.17	0.26	0.01
siz_102__141_no	0.06	0.01	**0.72**	0.03	-0.01	0.11
siz_102__141_sa	0.00	0.00	**0.65**	-0.03	-0.02	-0.01
siz_512__143_ao	-0.19	0.01	0.14	**0.49**	**0.39**	0.07
siz_204__139_no	0.04	0.02	0.02	**0.35**	-0.01	-0.12
sgp_32__29_ao	0.04	0.29	-0.03	**0.29**	-0.12	0.05
jou_hea__36_no	-0.05	0.16	-0.01	**0.26**	-0.03	-0.05
skb_hea_c_91_ao	-0.02	0.02	0.20	0.11	**0.57**	0.01
fil___108_ao	0.27	0.10	0.06	0.10	**0.57**	0.01
fil___108_no	0.05	0.05	-0.02	-0.04	**0.24**	0.01
ext_ino_3_33_ao	-0.02	0.01	0.03	-0.02	0.02	**0.68**
siz_204__139_sa	0.02	0.01	0.02	0.09	0.02	0.02
sgp_16__30_ao	0.02	0.17	0.01	0.09	0.05	0.00

Table 39 – Factor loadings of variables tracked by /proc/stat ($n1p1c1aX$, split HPCC).

Variable	Factor1	Factor2	Factor3	Factor4	Factor5	Factor6
user	**1.00**	0.00	-0.03	0.00	-0.04	0.00
Watts	**0.97**	0.00	-0.13	0.00	-0.04	0.00
procs_r	**0.33**	0.01	-0.01	0.00	-0.01	0.00
int100	0.00	**0.99**	0.01	-0.09	-0.04	-0.02
int1	0.00	**0.98**	0.01	0.21	0.01	-0.01
ctxt	0.01	**0.70**	0.01	0.03	0.01	0.02
procs	-0.01	**0.50**	0.01	-0.04	0.01	-0.02
sirq	0.01	**0.35**	0.00	-0.02	0.04	0.00
sys	0.02	0.01	**1.00**	0.00	0.02	0.00
int16	0.00	0.08	0.00	**0.99**	0.00	0.00
int76	0.04	0.07	-0.01	-0.01	**0.99**	-0.03
int2	0.00	0.03	0.00	0.00	0.03	**1.00**
hirq	0.00	0.10	0.00	0.02	0.02	0.01
iow	0.01	0.00	0.00	0.00	0.12	0.00
sysi	**-0.94**	0.00	-0.32	0.00	0.03	0.00
procs_b	-0.01	0.00	0.00	0.00	0.02	0.00

Table 40 – Factor loadings of variables tracked by /proc/vmstat ($n1p1c1aX$, split HPCC).

Variable	Factor1	Factor2	Factor3	Factor4	Factor5	Factor6
nr_anon_pages	**0.99**	-0.01	0.06	0.01	0.02	-0.09
nr_page_table_pages	**0.99**	-0.01	0.06	0.01	0.02	-0.08
Watts	**0.92**	0.05	0.07	-0.01	0.01	-0.03
nr_mapped	**0.74**	0.04	0.13	-0.01	0.00	**0.65**
pgalloc_normal	-0.02	**0.98**	0.03	-0.16	0.04	0.00
numa_hit	0.00	**0.97**	0.03	**0.23**	0.03	0.00
pgfault	0.00	**0.97**	0.03	**0.23**	0.03	0.00
nr_slab	0.11	-0.01	**0.99**	0.02	0.03	0.04
nr_file_pages	-0.04	-0.03	**0.94**	0.02	-0.04	-0.05
pgalloc_dma32	0.04	0.21	-0.01	**0.97**	0.00	0.00
pgactivate	-0.01	-0.01	0.03	0.01	**0.52**	0.01
nr_writeback	0.00	-0.01	0.01	0.01	**0.36**	0.00
nr_unstable	-0.01	-0.01	0.00	0.00	**0.23**	0.01
pgfree	-0.03	0.00	0.01	0.00	0.00	**0.06**
nr_dirty	0.01	0.01	0.06	-0.01	0.02	0.02
numa_interleave	-0.10	0.01	0.01	-0.01	0.02	0.02
pgpgout	0.00	0.02	0.01	-0.01	0.05	0.00

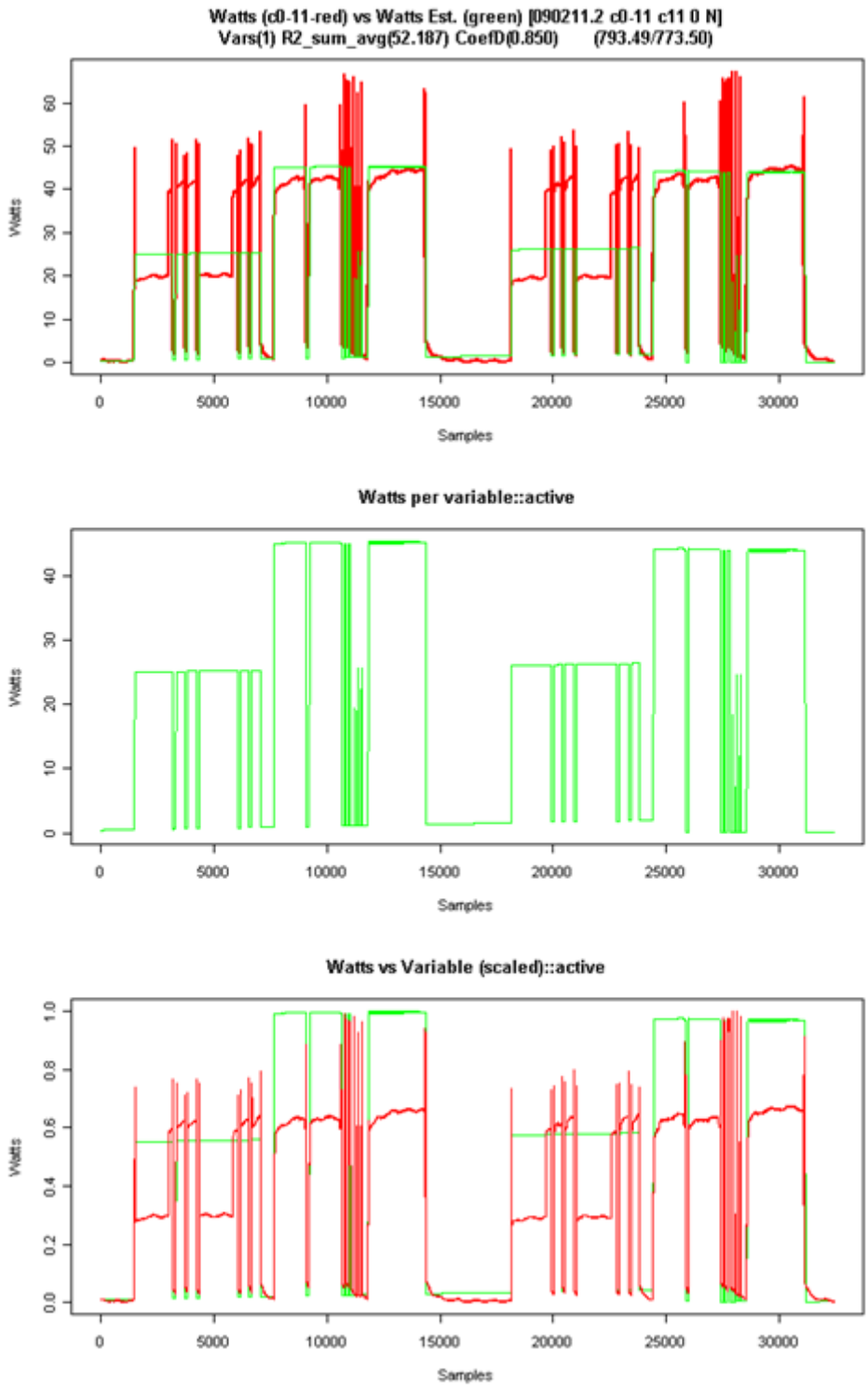


Figure 61 – /proc/meminfo.*active* single variable model (*n*1*p*1*c*1*aX*, split HPCC).

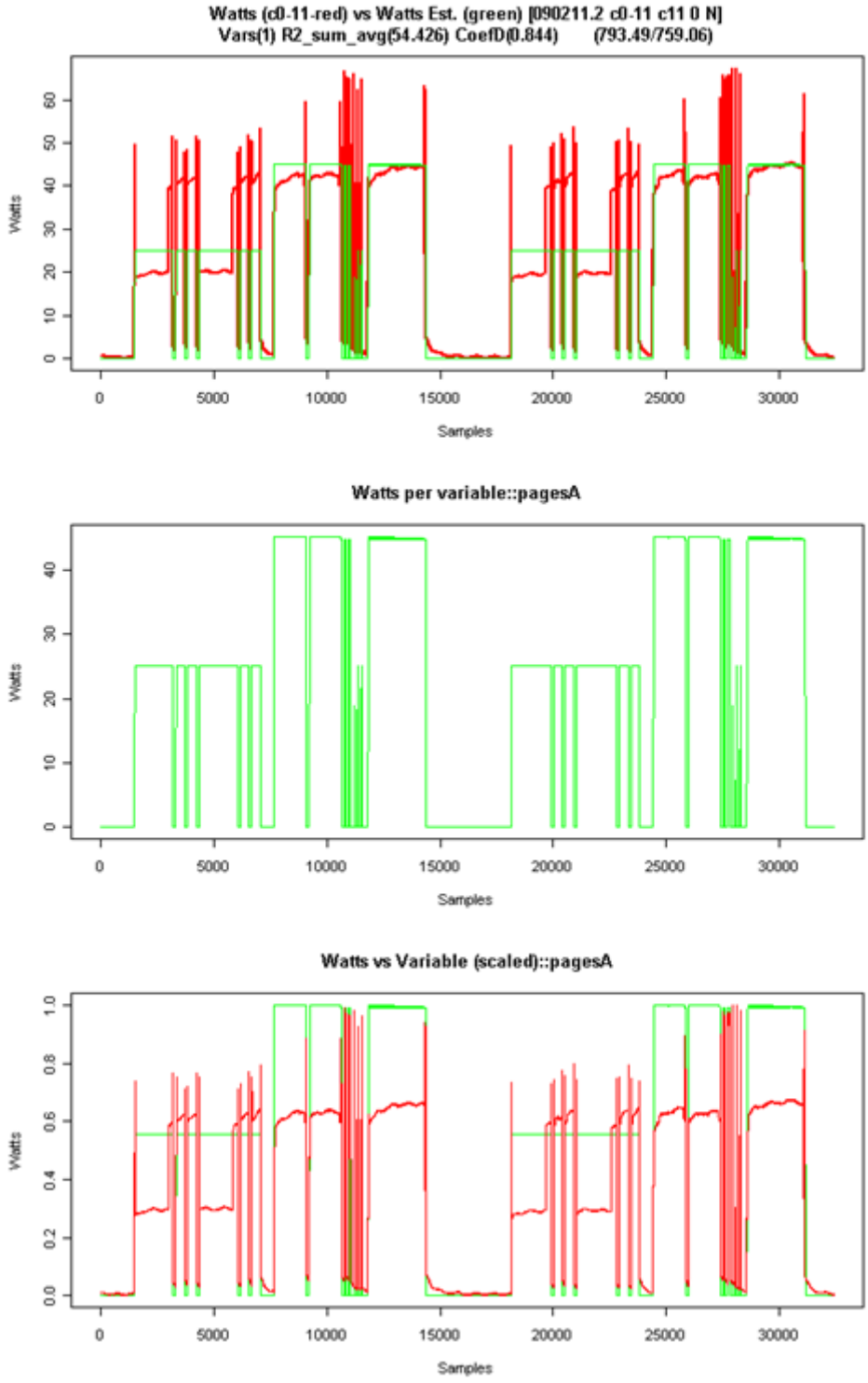


Figure 62 – /proc/meminfo.*pagesA* single variable model (*n*1*p*1*c*1*aX*, split HPCC).

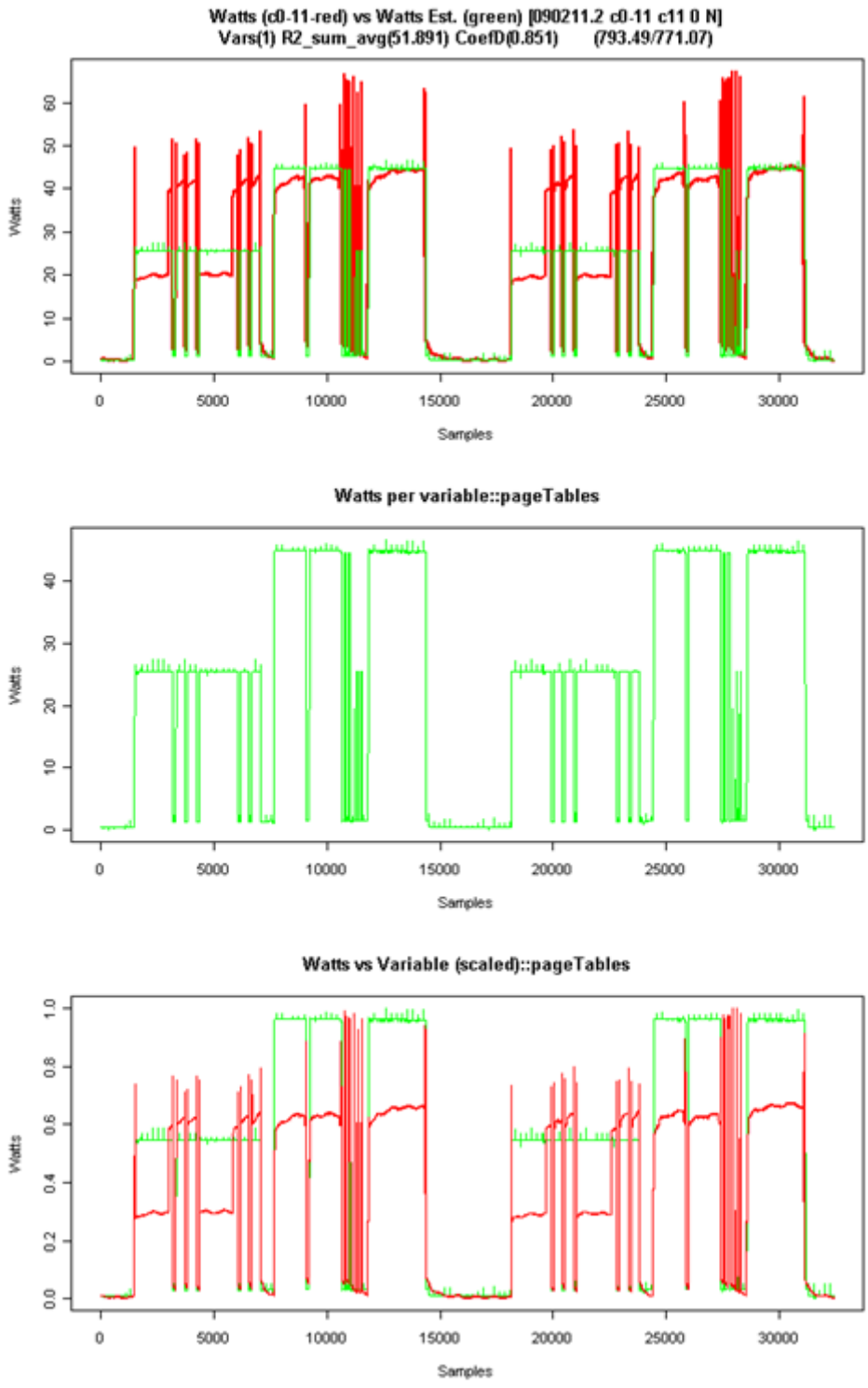


Figure 63 – /proc/meminfo.*pageTables* single variable model (*n*1*p*1*c*1*aX*, split HPCC).

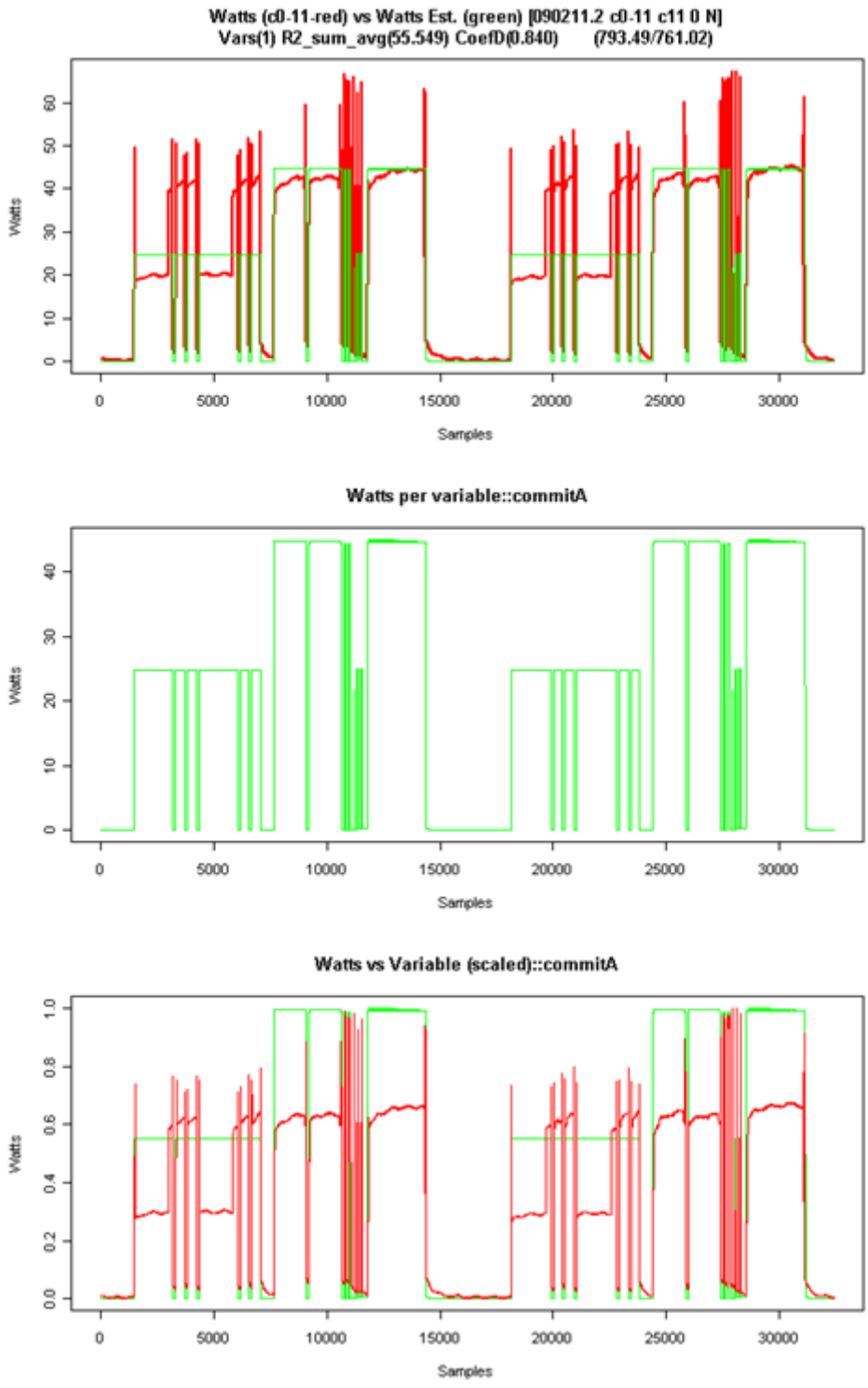


Figure 64 – /proc/meminfo.*commitA* single variable model (*n*1*p*1*c*1*aX*, split HPCC).

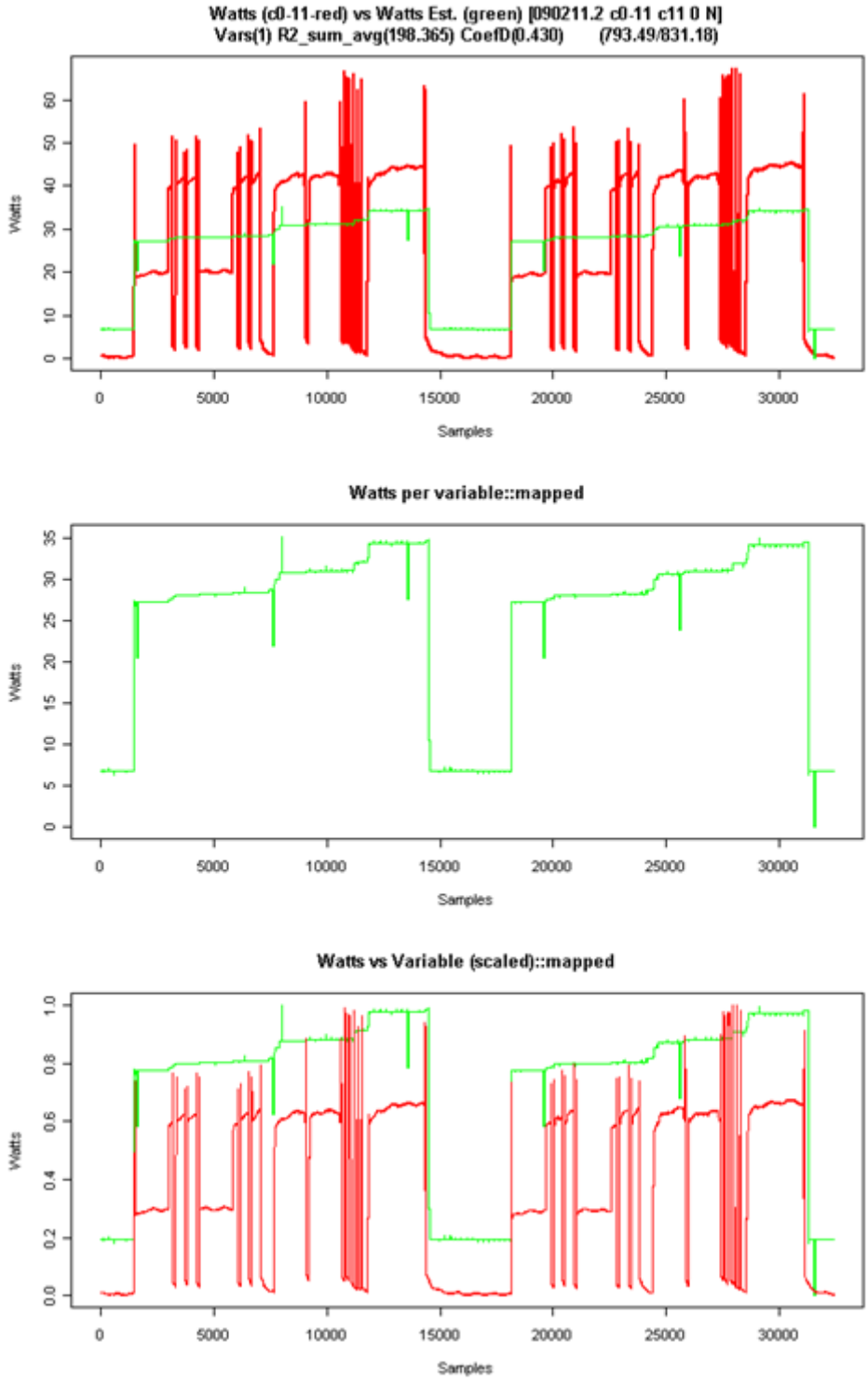


Figure 65 – /proc/meminfo.*mapped* single variable model (*n*1*p*1*c*1*aX*, split HPCC).

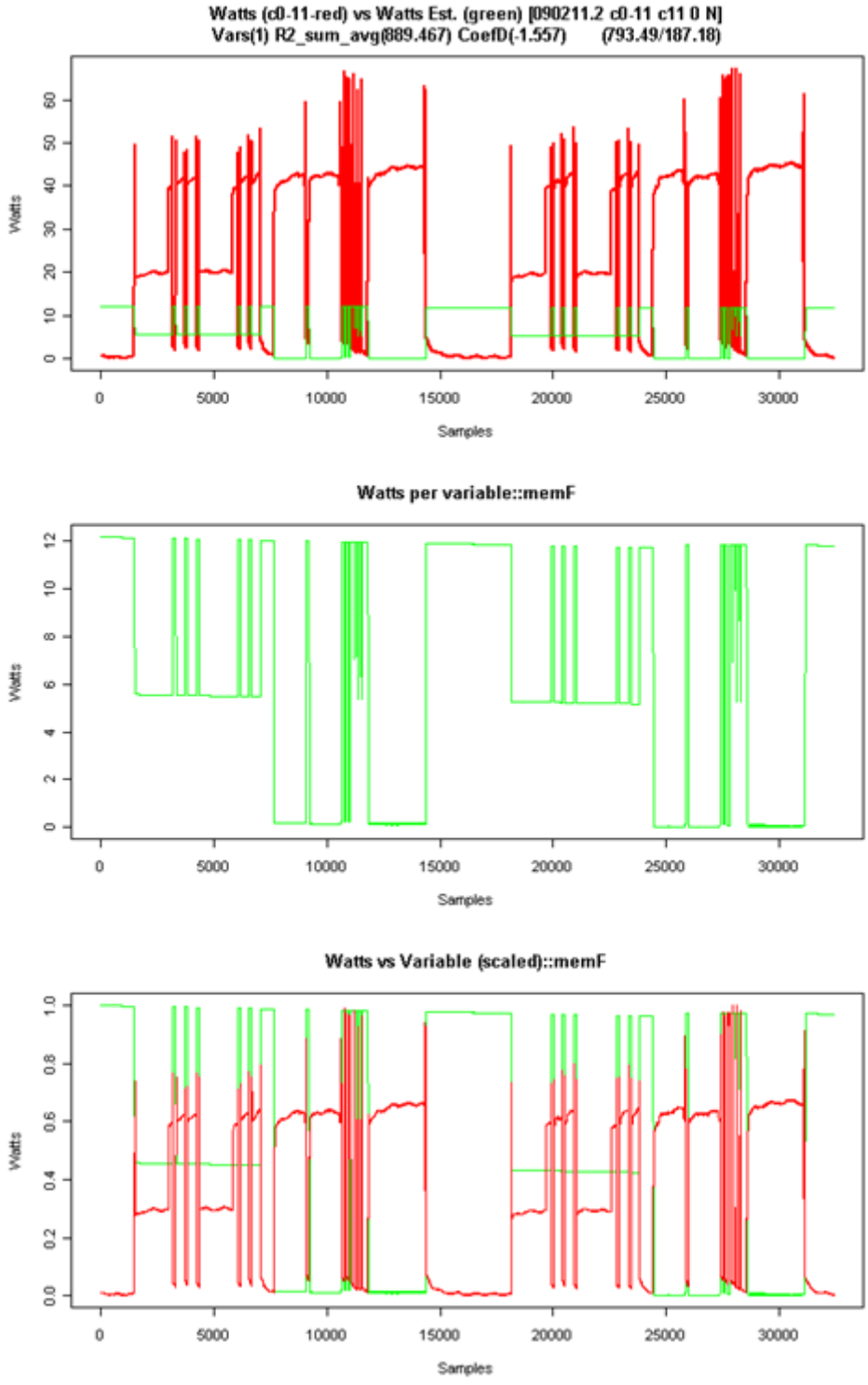


Figure 66 – /proc/meminfo.*memF* single variable model (*n*1*p*1*c*1*aX*, split HPCC).

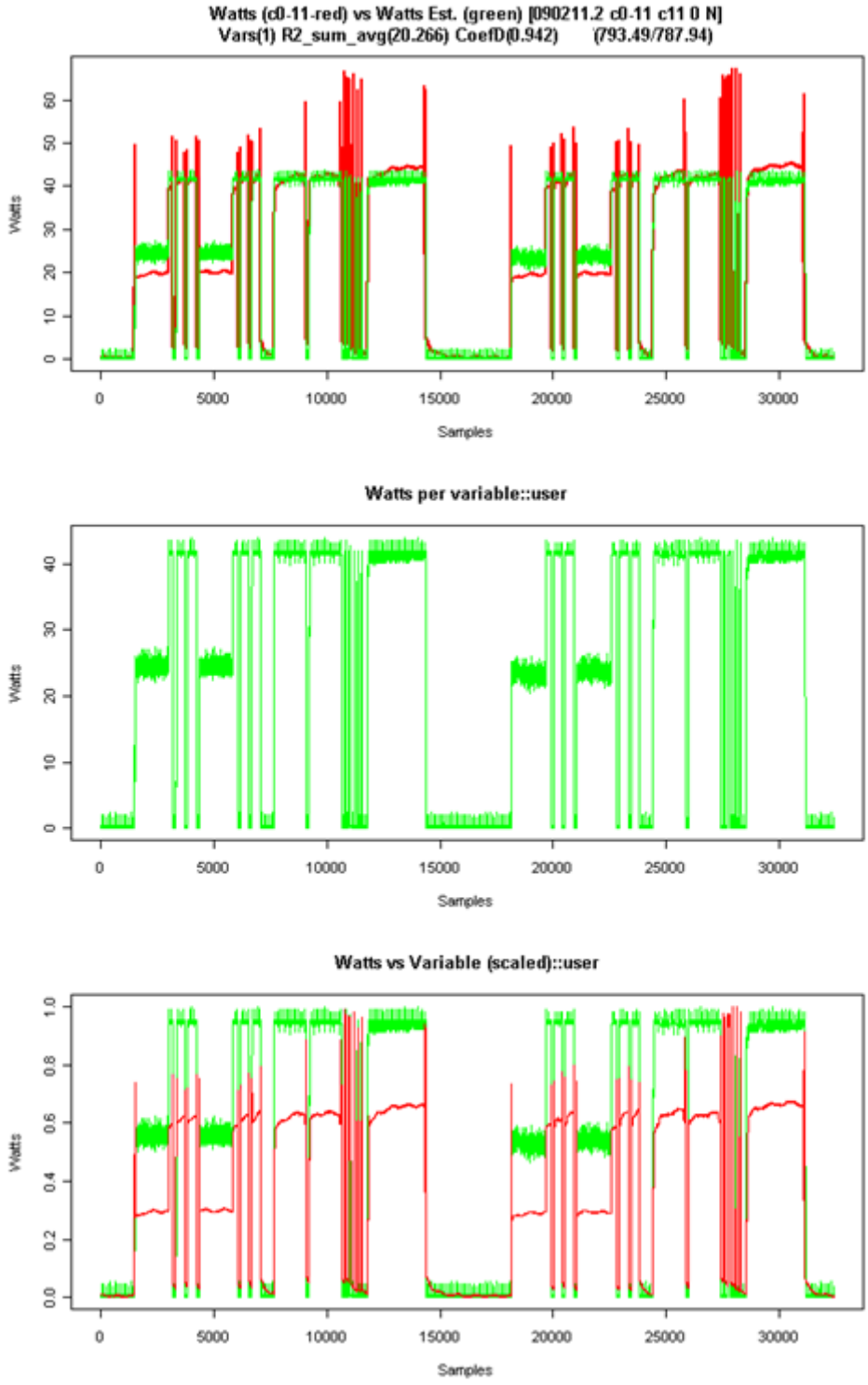


Figure 67 – /proc/stat.*user* single variable model (*n*1*p*1*c*1*aX*, split HPCC).

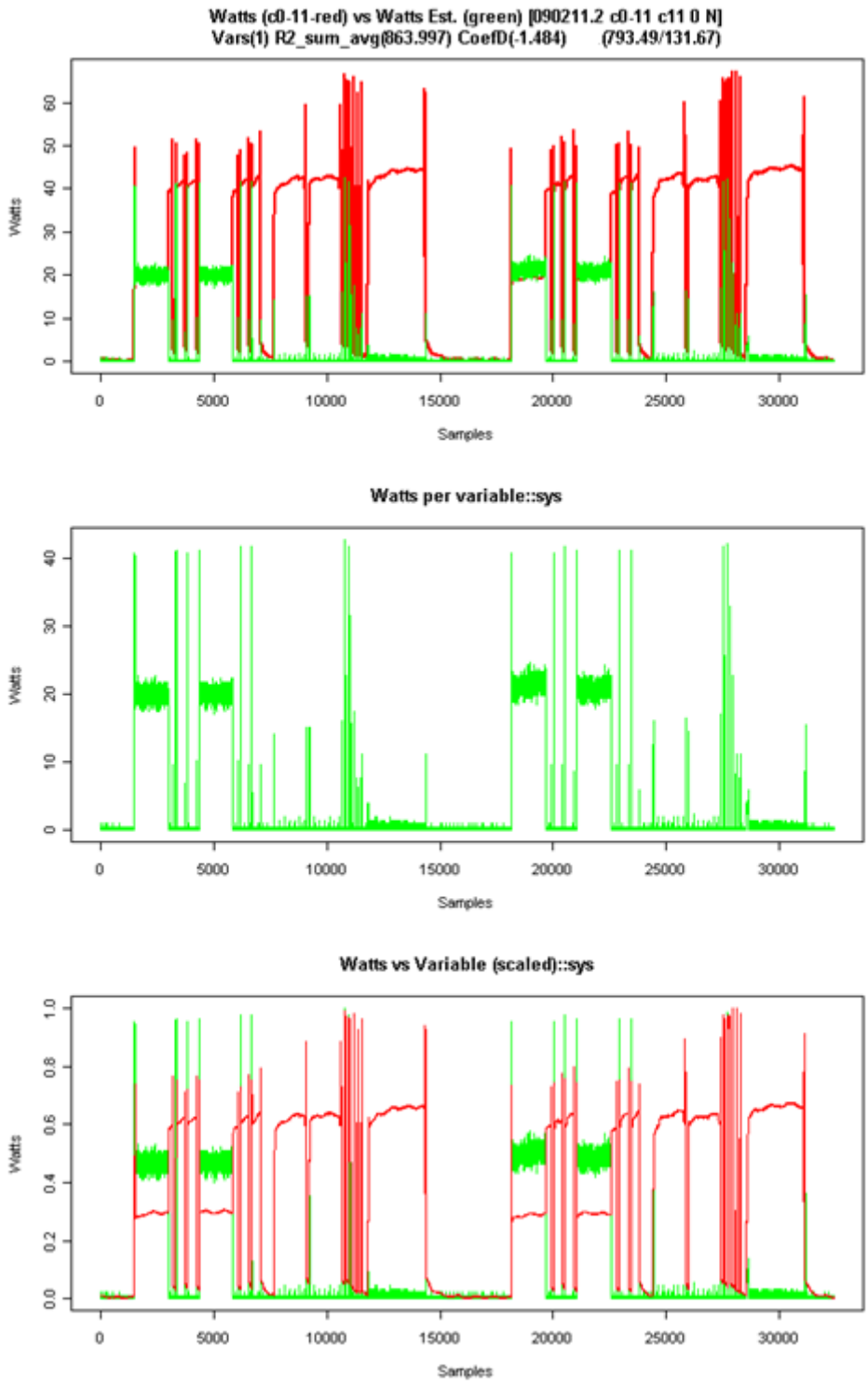


Figure 68 – /proc/stat.*sys* single variable model (*n*1*p*1*c*1*aX*, split HPCC).

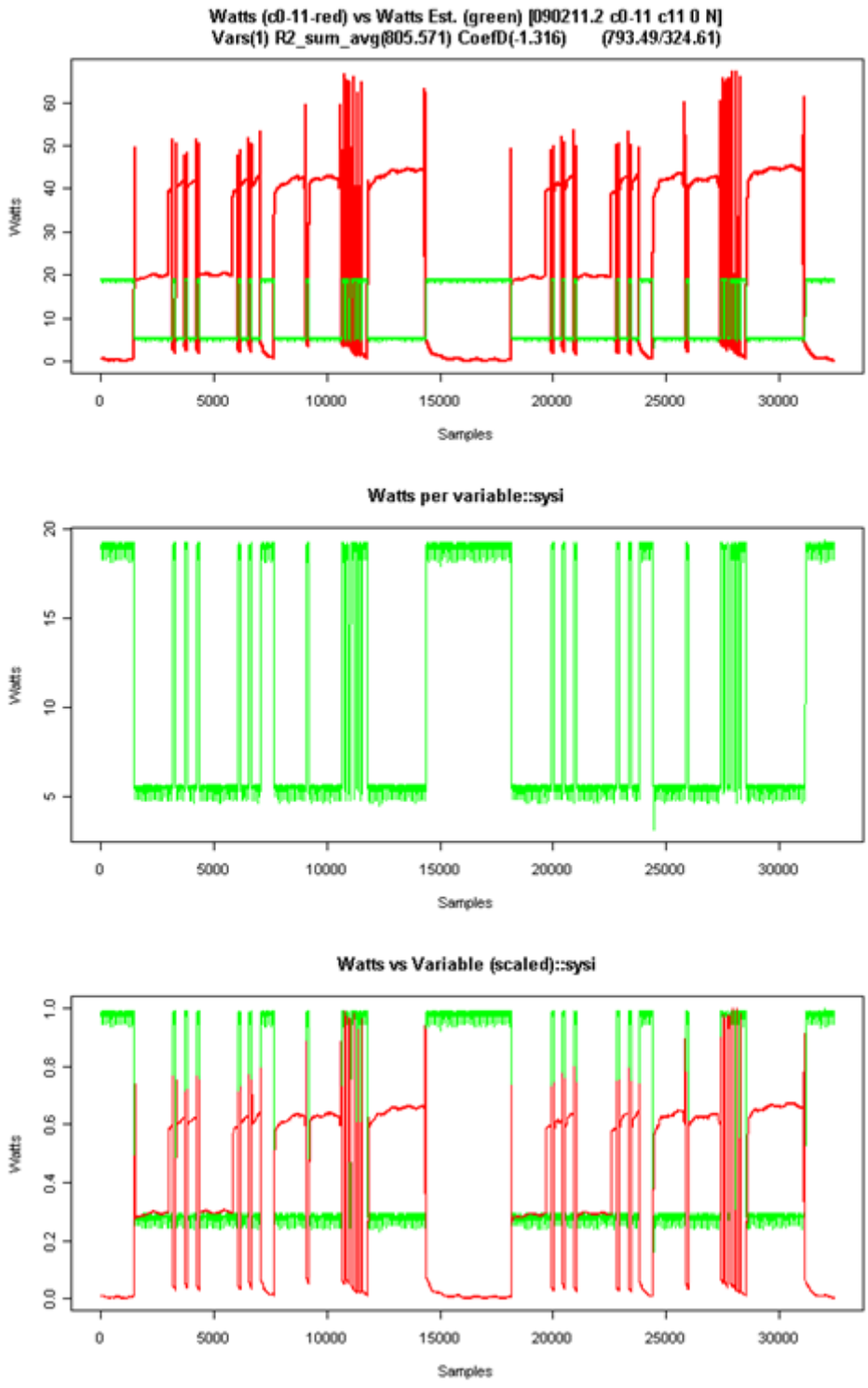


Figure 69 – /proc/stat.*sysi* single variable model (*n*1*p*1*c*1*aX*, split HPCC).

Table 41 – Factor analysis of meminfo and stat ($n1p1c1aX$, Split HPCC).

Variable	R2_sum_avg	CoefD	Est. kWs	% Err
active	52.19	0.85	773.50	2.52
pagesA	54.43	0.84	759.06	4.34
pageTables	51.89	0.85	771.07	2.83
commitA	55.55	0.84	761.02	4.09
(mapped)	198.37	0.43	831.18	4.75
memF	889.47	-1.56	187.18	76.41
user	20.27	0.94	787.94	0.70
sys	864.00	-1.48	131.67	83.41
sysi	805.57	-1.32	324.61	59.09
Tot. kWs			**793.49**	

Table 42 – Clusters identified for *n*1*p*1*c*1*aX*, Split HPCC.

Time processor spent executing in user mode

- /proc/stat.*user*
- /proc/meminfo.*active*
- /proc/meminfo.*pagesA*
- /proc/meminfo.*pageTables*
- /proc/meminfo.*commitA*
- /proc/meminfo.*mapped* (the least related to power consumption)
- /proc/interrupts.*nmi_CPUX* (where *X* is a CPU core)
- /proc/schedstat.*cpuX_4* (where *X* is a CPU core)
- /proc/schedstat.*cpuX_9* (where *X* is a CPU core)

Time processor spent executing in system mode

- /proc/stat.*sys*

Time processor spent idling

- /proc/stat.*sysi*
- /proc/schedstat.*cpu0_6*
- /proc/buddyinfo.*dma32_1*
- /proc/buddyinfo.*dma32_2*
- /proc/buddyinfo.*normal_X* (where *X* is CPU core 2 through 7)

Ethernet interface interrupts

- /proc/stat.*int1*
- /proc/stat.*int100*
- /proc/interrupts.*eth0_cpu6*
- /proc/stat.*ctxt*
- /proc/schedstat.*cpu6_6*

Table 42 – Clusters identified for *n*1*p*1*c*1*aX*, Split HPCC (Continued).

Ethernet interface transmitted and received packets or bytes

- /proc/net/dev.*eth0_Rx_B*
- /proc/net/dev.eth0_*Tx_B*
- /proc/net/dev.*eth0_Rx_Pkt*
- /proc/net/dev.*eth0_Tx_Pkt*
- /proc/slabinfo.*skb_fcl_c_90_ao*
- /proc/slabinfo.*skb_hea_c_91_ao*
- /proc/slabinfo.*siz_102__141_ao*

Disk utilization (sda)

- /proc/stat.*int76*
- /proc/interrupts.*usb_4_CPU1*
- /proc/diskstats.*sda_17_com_w*
- /proc/diskstats.*sda_17_sec_w*
- /proc/diskstats.*sda1_18_sec_r*
- /proc/diskstats.*sda_17_msec_w*
- /proc/diskstats.*sda_17_msec_io_w*
- /proc/vmstat.*nr_dirty*
- /proc/vmstat.*pgpgout*

Memory (page utilization)

- /proc/meminfo.*active*
- /proc/meminfo.*pagesA*
- /proc/meminfo.*pageTables*
- /proc/meminfo.*commitA*
- /proc/meminfo.*mapped*
- /proc/vmstat.*nr_anon_pages*
- /proc/vmstat.*nr_page_table_pages*
- /proc/vmstat.*nr_mapped*

Table 42 – Clusters identified for *n*1*p*1*c*1*aX*, Split HPCC (Continued).

Memory (Page faults)

- /proc/vmstat.*numa_hit*
- /proc/vmstat.*numa_local*
- /proc/vmstat.*pgalloc_dma32*
- /proc/vmstat.*pgfault*
- /proc/vmstat.*pgalloc_normal*

Memory (Cache)

- /proc/meminfo.*cache*
- /proc/meminfo.*slab*
- /proc/meminfo.*buf*
- /proc/vmstat.*nr_file_pages*
- /proc/vmstat.*nr_slab*

Table 43 – Factor loadings for /proc/buddyinfo (*n*1*p*1*c*1*aX* and *n*20*p*40*c*160*aX*, Split HPCC).

	cores:1, c0-11			cores:160, c0-11			cores:160, c0-12	
Variable	Factor1	Factor2		Factor1	Factor2		Factor1	Factor2
dma32_0	**0.73**	0.22		**0.97**	0.25		**0.97**	0.23
dma32_1	**0.94**	0.30		**0.97**	0.25		**0.97**	0.23
dma32_2	**0.95**	0.29		**0.97**	0.25		**0.97**	0.22
dma32_3	**0.96**	0.28		**0.94**	0.27		**0.97**	0.22
dma32_4	**0.96**	0.27		**0.95**	0.24		**0.95**	0.23
dma32_5	**0.96**	0.27		**0.88**	0.22		**0.97**	0.19
dma32_6	**0.96**	0.27		**0.65**	0.01		**0.92**	0.19
dma32_7	**0.95**	0.28		**0.43**	-0.09		**0.86**	0.13
normal_0	0.03	0.14		0.13	**0.78**		0.15	**0.79**
normal_1	0.12	**0.45**		0.14	**0.99**		0.16	**0.99**
normal_2	0.26	**0.95**		0.14	**0.99**		0.16	**0.99**
normal_3	0.26	**0.94**		0.14	**0.99**		0.16	**0.98**
normal_4	0.26	**0.94**		0.15	**0.96**		0.17	**0.95**
normal_5	0.27	**0.95**		**0.87**	**0.30**		**0.91**	**0.27**
normal_6	0.27	**0.94**		**0.92**	0.19		**0.92**	0.19
normal_7	0.38	**0.78**		**0.88**	0.07		**0.90**	0.06
Watts	**-0.54**	-0.75		**-0.21**	-0.93		**-0.27**	-0.93

Table 44 – Factor loadings for /proc/diskstats (*n*1*p*1*c*1*aX* and *n*20*p*40*c*160*aX*, Split HPCC).

	cores:1, c0-11			cores:160, c0-11			cores:160, c0-12	
Variable	Factor1	Factor2		Factor1	Factor2		Factor1	Factor2
sda_17_com_w	**0.82**	0.23		**0.86**	0.20		**0.84**	0.18
sda_17_mer_w	**0.91**	-0.05		**1.00**	0.03		**1.00**	0.03
sda_17_sec_w	**0.99**	0.09		**1.00**	0.06		**1.00**	0.05
sda_17_msec_w	0.29	**0.92**		**0.85**	0.00		**0.95**	0.03
sda_17_msec_io	0.21	**0.86**		**0.71**	0.05		**0.71**	0.15
sda_17_msec_io_w	0.27	**0.96**		**0.85**	0.00		**0.95**	0.03
sda1_18_sec_r	**0.93**	0.11		**1.00**	0.02		**1.00**	0.02
sda4_21_sec_r	**0.67**	-0.01		0.04	**1.00**		0.04	**1.00**
Watts	**0.00**	0.01		**0.12**	-0.01		**0.11**	-0.01

Table 45 – Factor loadings for /proc/interrupts (*n*1*p*1*c*1*aX* and *n*20*p*40*c*160*aX*, Split HPCC).

	cores:1, c0-11			cores:160, c0-11			cores:160, c0-12	
Variable	Factor1	Factor2		Factor1	Factor2		Factor1	Factor2
ide0_CPU4	0.00	0.00		0.41	0.12		0.41	0.09
ide0_CPU4	-	-		-0.36	-0.10		-0.36	-0.08
ide0_CPU5	-	-		-	-		0.01	0.02
usb_4_CPU1	-	-		0.16	0.06		0.16	0.06
usb_4_CPU4	0.00	-0.01		-0.04	-0.06		-0.05	-0.05
usb_4_CPU5	-	-		-0.02	-0.03		-0.02	-0.02
timer_CPU0	0.00	0.00		0.01	0.01		0.00	0.03
eth0_CPU6	0.00	0.00		0.01	0.05		0.02	0.03
eth0_CPU7	-	-		0.57	0.57		0.68	0.39
nmi_CPU0	**0.87**	-0.24		**0.88**	0.46		**0.94**	0.32
nmi_CPU1	0.01	-0.01		**0.88**	0.46		**0.94**	0.32
nmi_CPU2	-0.02	-0.05		**0.88**	0.46		**0.94**	0.32
nmi_CPU3	0.00	-0.01		**0.88**	0.46		**0.94**	0.32
nmi_CPU4	-0.04	-0.01		**0.88**	0.46		**0.94**	0.32
nmi_CPU5	-0.01	-0.02		**0.88**	0.46		**0.94**	0.33
nmi_CPU6	-0.01	0.00		**0.88**	0.46		**0.94**	0.33
nmi_CPU7	-0.04	0.99		**0.88**	0.46		**0.94**	0.32
Watts	**0.80**	0.39		**0.95**	0.29		**0.99**	0.12

Table 46 – Factor loadings for /proc/loadavg (*n*1*p*1*c*1*aX* and *n*20*p*40*c*160*aX*, Split HPCC).

	cores:1, c0-11			cores:160, c0-11			cores:160, c0-12	
Variable	Factor1	Factor2		Factor1	Factor2		Factor1	Factor2
load_avg_1	**0.92**	0.38		**0.79**	0.57		**0.82**	0.45
load_avg_5	**0.70**	0.68		0.55	0.83		0.58	0.73
load_avg_15	0.38	0.92		0.27	0.89		0.29	0.95
kse_r	0.21	0.13		**0.88**	0.30		**0.86**	0.33
kse_n	0.51	0.51		0.48	0.48		0.51	0.51
Watts	**0.69**	0.37		**0.90**	0.38		**0.91**	0.32

Table 47 – Factor loadings for (*n*1*p*1*c*1*aX* and *n*20*p*40*c*160*aX*, Split HPCC).

	cores:1, c0-11			cores:160, c0-11			cores:160, c0-12	
Variable	Factor1	Factor2		Factor1	Factor2		Factor1	Factor2
memF	-0.98	-0.06		-1.00	0.01		-0.96	-0.27
buf	-0.08	0.69		0.05	**1.00**		0.03	-0.09
cache	-0.01	**0.96**		0.07	**1.00**		0.05	-0.07
active	**0.99**	0.05		**1.00**	-0.01		**0.96**	0.26
inactive	0.14	0.27		0.39	-0.43		0.15	**0.87**
dirty	0.01	0.05		**0.79**	0.01		**0.78**	0.14
writeb	0.00	0.00		-0.03	0.02		0.05	0.03
pagesA	**0.99**	0.04		**1.00**	-0.06		**0.96**	0.27
mapped	0.63	0.16		0.62	-0.06		0.39	**0.89**
slab	0.10	**0.90**		**0.64**	-0.04		**0.40**	**0.90**
pageTables	**0.98**	0.04		**0.97**	-0.08		**0.87**	0.49
nfs	-0.01	0.00		0.02	-0.01		0.03	0.00
commitA	**0.98**	0.04		**1.00**	-0.05		**0.96**	0.28
Watts	**0.91**	0.06		**0.88**	-0.15		**0.77**	0.42

Table 48 – Factor loadings for /proc/stat, interrupts, and net/dev (*n*1*p*1*c*1*aX* and *n*20*p*40*c*160*aX*).

	cores:1, c0-11			cores:160, c0-11			cores:160, c0-12	
Variable	Factor1	Factor2		Factor1	Factor2		Factor1	Factor2
int1	**0.94**	0.12		**0.98**	0.19		**1.00**	-0.01
int100	**0.98**	0.12		**0.98**	0.19		**1.00**	-0.01
ctxt	-	-		0.00	0.00		-0.01	0.17
eth0_CPU6	**0.98**	0.12		0.01	0.01		0.04	1.00
eth0_CPU7	-	-		**0.98**	0.19		**1.00**	-0.07
lo_Tx_Pkt	0.09	0.00		-0.01	0.00		-0.01	0.04
eth0_Rx_Pkt	0.12	**0.98**		**0.29**	**0.95**		**0.48**	0.00
eth0_Tx_Pkt	0.10	**0.99**		**0.29**	**0.95**		**0.48**	0.00
Watts	0.00	0.00		**0.68**	**0.25**		**0.72**	-0.01

Table 49 – Factor loadings for /proc/schedstat ($n1p1c1aX$ and $n20p40c160aX$, Split HPCC).

	cores:1, c0-11			cores:160, c0-11			cores:160, c0-12	
Variable	Factor1	Factor2		Factor1	Factor2		Factor1	Factor2
cpu0_4	**0.90**	0.01		0.26	**0.96**		0.35	**0.93**
cpu1_4	0.05	**0.82**		0.52	**0.71**		0.37	**0.92**
cpu2_4	0.00	0.17		0.27	**0.95**		0.33	**0.94**
cpu3_4	0.03	**0.98**		0.60	**0.50**		0.70	**0.70**
cpu4_4	0.00	0.00		0.25	**0.96**		**0.92**	**0.35**
cpu5_4	-0.09	**0.75**		0.31	**0.92**		0.73	**0.65**
cpu6_4	-0.01	0.50		0.21	**0.97**		0.39	**0.91**
cpu7_4	0.08	0.28		0.30	**0.93**		**0.92**	**0.35**
cpu0_6	-0.85	-0.01		-0.85	-0.25		-0.84	-0.32
cpu1_6	0.24	0.02		-0.22	-0.06		-0.55	-0.21
cpu2_6	-0.07	0.05		-0.84	-0.24		-0.84	-0.32
cpu3_6	-0.11	-0.01		-0.32	-0.10		-0.18	-0.08
cpu4_6	0.06	0.03		-0.84	-0.25		-0.83	-0.32
cpu5_6	0.10	0.00		-0.27	-0.09		-0.13	-0.06
cpu6_6	0.04	0.01		-0.59	-0.17		-0.59	-0.20
cpu7_6	0.05	0.02		-0.21	-0.06		-0.34	-0.13
cpu0_9	**0.90**	0.01		**0.93**	0.27		**0.92**	**0.34**
cpu1_9	-0.08	-0.03		**0.93**	0.27		**0.92**	**0.34**
cpu2_9	-0.03	-0.06		**0.93**	0.27		**0.92**	**0.34**
cpu3_9	-0.07	0.05		**0.93**	0.27		**0.92**	**0.34**
cpu4_9	0.00	0.01		**0.93**	0.27		**0.92**	**0.34**
cpu5_9	-0.27	0.02		**0.93**	0.27		**0.92**	**0.34**
cpu6_9	-0.08	0.00		**0.93**	0.27		**0.92**	**0.35**
cpu7_9	0.06	-0.02		**0.93**	0.27		**0.92**	**0.35**
Watts	**0.62**	-0.02		**0.92**	0.27		**0.90**	**0.34**

Table 50 – Factor loadings for /proc/stat (*n*1*p*1*c*1*aX* and *n*20*p*40*c*160*aX*, Split HPCC).

	cores:1, c0-11			cores:160, c0-11			cores:160, c0-12	
Variable	Factor1	Factor2		Factor1	Factor2		Factor1	Factor2
user	**1.00**	0.00		0.13	0.70		-0.17	0.68
sys	0.02	0.01		**0.92**	0.15		**0.77**	0.53
sysi	-0.94	0.00		-0.91	-0.40		-0.66	-0.75
iow	0.01	0.00		-0.03	-0.01		-0.01	-0.01
hirq	0.00	0.10		**0.93**	-0.25		**0.95**	0.16
sirq	0.01	0.35		**0.83**	-0.20		**0.84**	0.16
int1	0.00	**0.98**		**0.96**	-0.27		**0.99**	0.16
int2	0.00	0.03		0.01	0.00		0.00	0.00
int16	0.00	0.08		0.00	0.00		0.00	0.00
int76	0.04	0.07		0.10	0.08		0.08	0.10
int100	0.00	**0.99**		**0.96**	-0.27		**0.99**	0.16
ctxt	0.01	**0.70**		0.00	-0.04		-0.01	-0.03
procs	-0.01	0.50		-0.01	0.00		0.00	-0.01
procs_r	0.33	0.01		**0.87**	0.38		0.62	0.68
procs_b	-0.01	0.00		-	-		0.02	0.00
Watts	**0.97**	0.00		**0.87**	0.47		0.61	0.77

Table 51 – Factor loadings for /proc/vmstat (*n*1*p*1*c*1*aX* and *n*20*p*40*c*160*aX*, Split HPCC).

	cores:1, c0-11			cores:160, c0-11			cores:160, c0-12	
Variable	**Factor1**	**Factor2**		**Factor1**	**Factor2**		**Factor1**	**Factor2**
nr_anon_pages	**0.99**	-0.01		**0.97**	0.07		**0.95**	0.08
nr_mapped	**0.74**	0.04		**0.77**	0.04		**0.78**	0.07
nr_file_pages	-0.04	-0.03		-0.01	0.02		-0.03	0.00
nr_slab	0.11	-0.01		**0.79**	0.03		**0.80**	0.07
nr_page_table_pages	**0.99**	-0.01		**1.00**	0.06		**0.99**	0.08
nr_dirty	0.01	0.01		**0.74**	0.08		**0.73**	0.07
nr_writeback	0.00	-0.01		-0.03	0.00		0.06	0.00
nr_unstable	-0.01	-0.01		0.02	0.00		0.03	0.00
numa_hit	0.00	**0.97**		0.10	**0.99**		0.12	**0.94**
numa_interleave	-0.10	0.01		-0.03	0.00		-0.03	0.02
pgpgout	0.00	0.02		0.15	0.01		0.14	0.01
pgalloc_dma32	0.04	0.21		0.32	0.43		0.33	0.21
pgalloc_normal	-0.02	**0.98**		-0.01	**0.95**		-0.02	**0.99**
pgfree	-0.03	0.00		0.11	0.06		0.16	0.05
pgactivate	-0.01	-0.01		0.00	0.01		0.00	0.01
pgfault	0.00	**0.97**		-0.03	**0.97**		0.01	**0.94**
pgmajfault	-	-		-0.01	0.00		-0.01	0.00
Watts	**0.92**	0.05		**0.90**	0.06		**0.88**	0.09

Table 52 – Factor loadings for stat, meminfo, vmstat, and slabinfo (*n*20*p*40*c*160*aXw11t2*).

Variable	Factor1	Factor2	Factor3	Factor4	Factor5	Factor6
int1	**0.98**	0.00	0.15	-0.03	-0.06	-0.03
int100	**0.98**	0.00	0.15	-0.03	-0.06	-0.03
hirq	**0.95**	0.01	0.15	-0.03	-0.06	-0.01
sirq	**0.84**	-0.01	0.17	0.04	-0.06	0.03
sys	**0.82**	0.34	0.25	0.05	-0.03	-0.20
procs_r	**0.71**	0.47	0.26	0.08	-0.10	0.19
Watts	**0.68**	0.56	0.31	0.06	0.00	0.24
siz_64__148_ao	0.56	0.29	0.52	0.08	0.03	0.10
crq_poo__45_ao	0.47	0.23	0.17	0.00	0.05	0.03
scs_cmd_c_26_ao	0.46	0.09	0.12	-0.02	0.03	0.02
pagesA	0.40	**0.84**	0.27	0.05	0.05	0.24
nr_anon_pages	0.40	**0.84**	0.27	0.05	0.05	0.24
commitA	0.40	**0.83**	0.29	0.06	0.05	0.24
user	-0.06	**0.47**	0.16	0.05	0.03	0.86
mapped	0.23	0.30	**0.91**	0.03	-0.03	0.06
slab	0.46	0.23	**0.85**	0.03	0.04	0.07
inactive	0.44	0.03	**0.79**	0.01	-0.37	0.02
vm_are_s_116_no	0.28	0.12	**0.76**	0.02	0.31	0.05
pgalloc_normal	0.04	-0.05	0.04	**0.99**	-0.01	0.04
numa_hit	0.01	0.12	0.05	**0.95**	0.01	0.01
buf	-0.19	0.01	0.05	0.01	0.98	0.02
pgfree	-0.02	0.09	0.09	0.05	0.01	0.05
nr_writeback	-0.06	-0.01	-0.01	-0.01	0.01	0.01
int76	0.07	0.12	0.04	0.00	0.00	-0.01
iow	-0.03	-0.02	-0.01	0.00	0.00	-0.01
numa_interleave	0.03	-0.06	0.00	0.01	-0.01	-0.01
nr_unstable	0.00	0.03	0.00	0.00	-0.01	-0.01
sysi	-0.75	-0.48	-0.30	-0.07	0.02	-0.19
memF	-0.39	-0.84	-0.28	-0.05	-0.09	-0.24

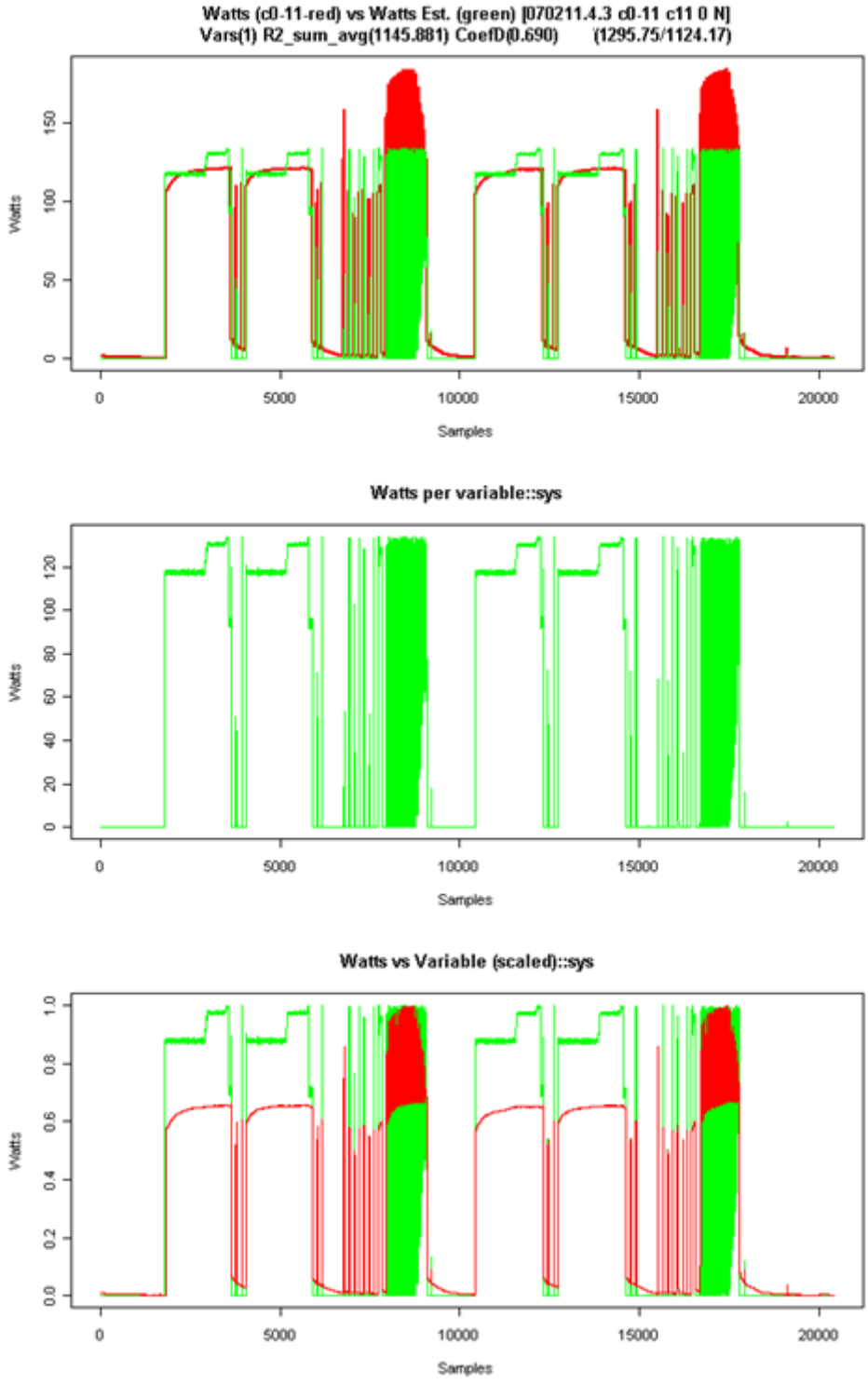


Figure 70 – /proc/stat.*sys* single variable model (*n*20*p*40*c*160*aX*, Split HPCC).

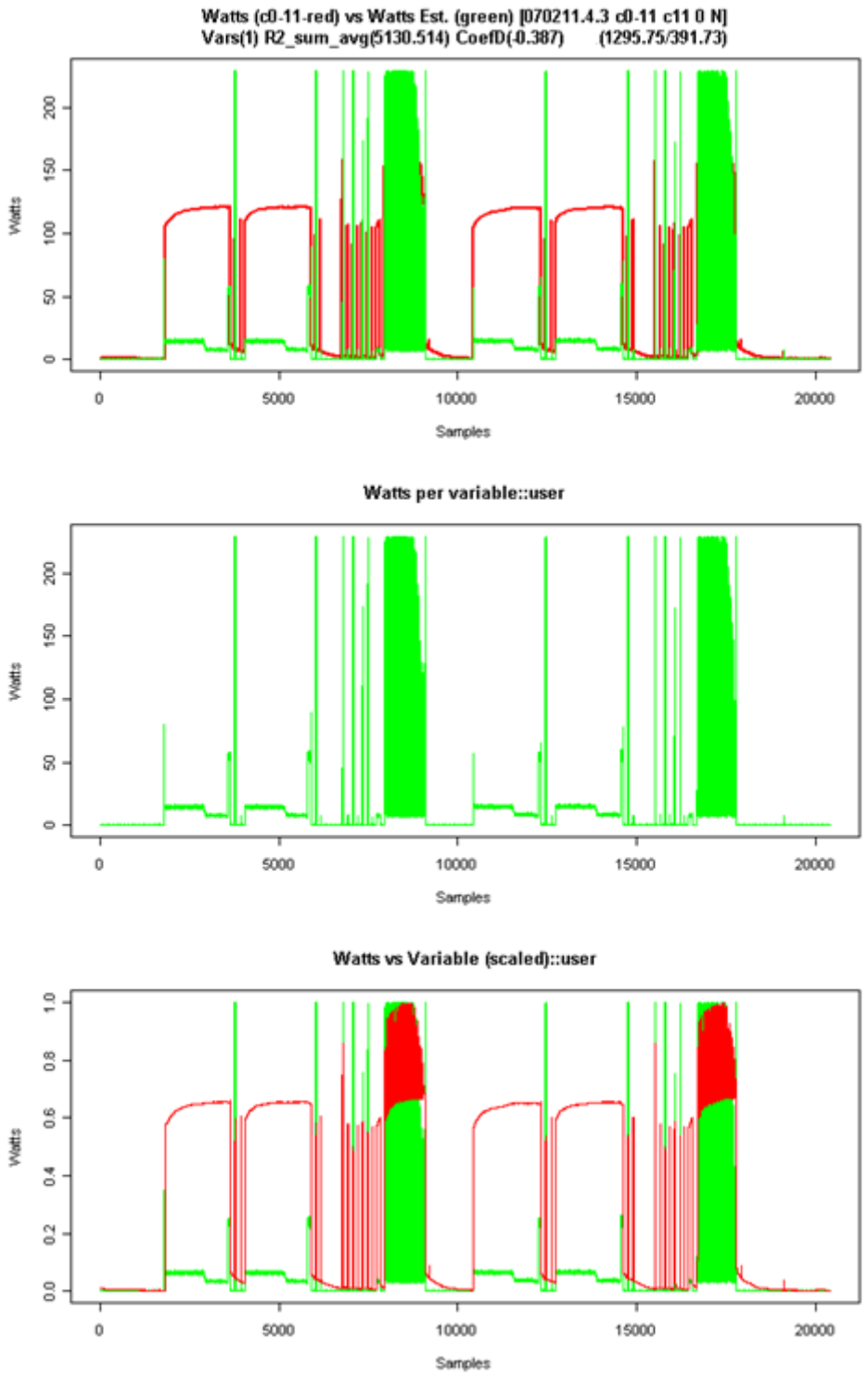


Figure 71 – /proc/stat.*user* single variable model (*n*20*p*40*c*160*aX*, Split HPCC).

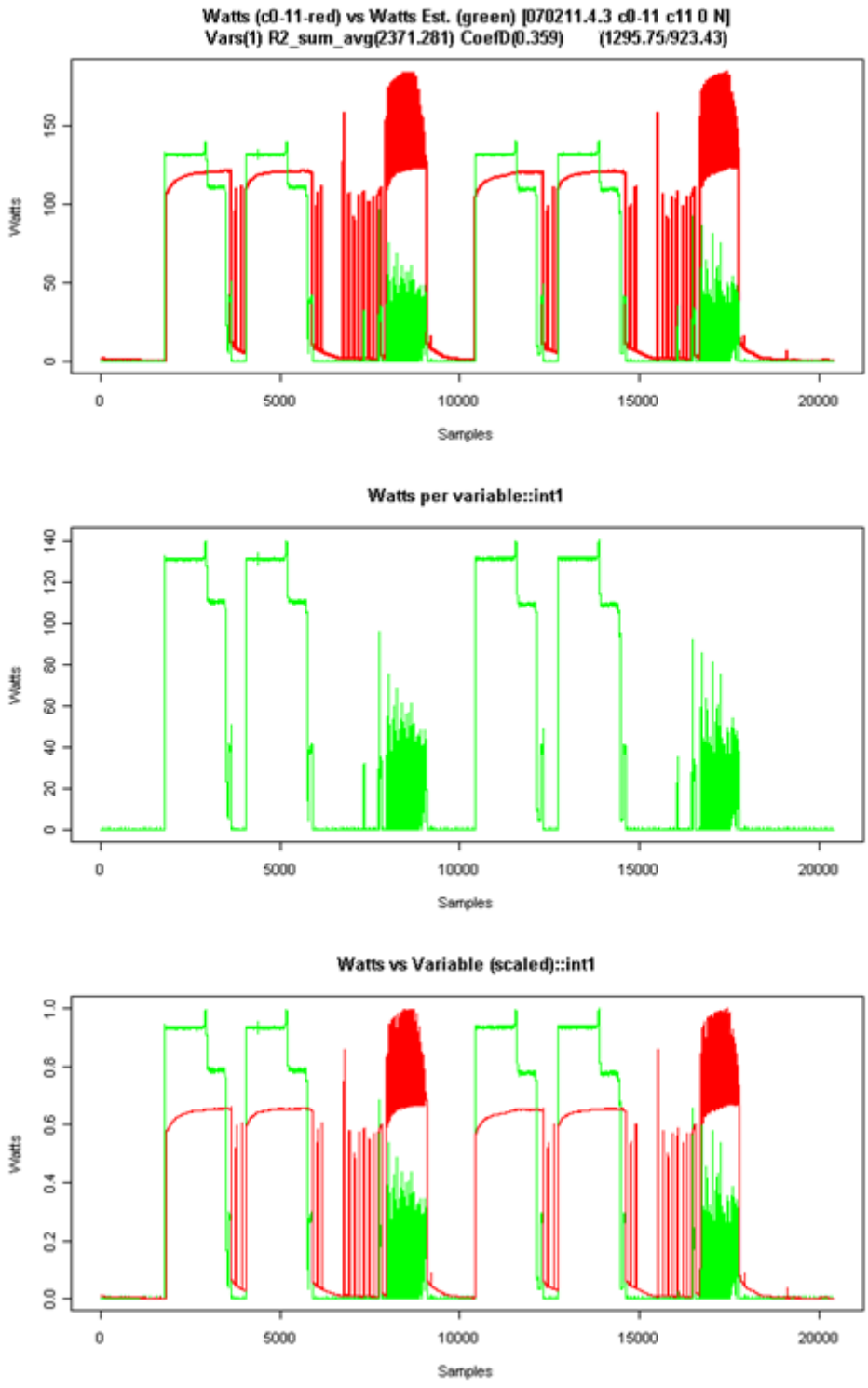


Figure 72 – /proc/stat.*int1* single variable model (*n*20*p*40*c*160*aX*, Split HPCC).

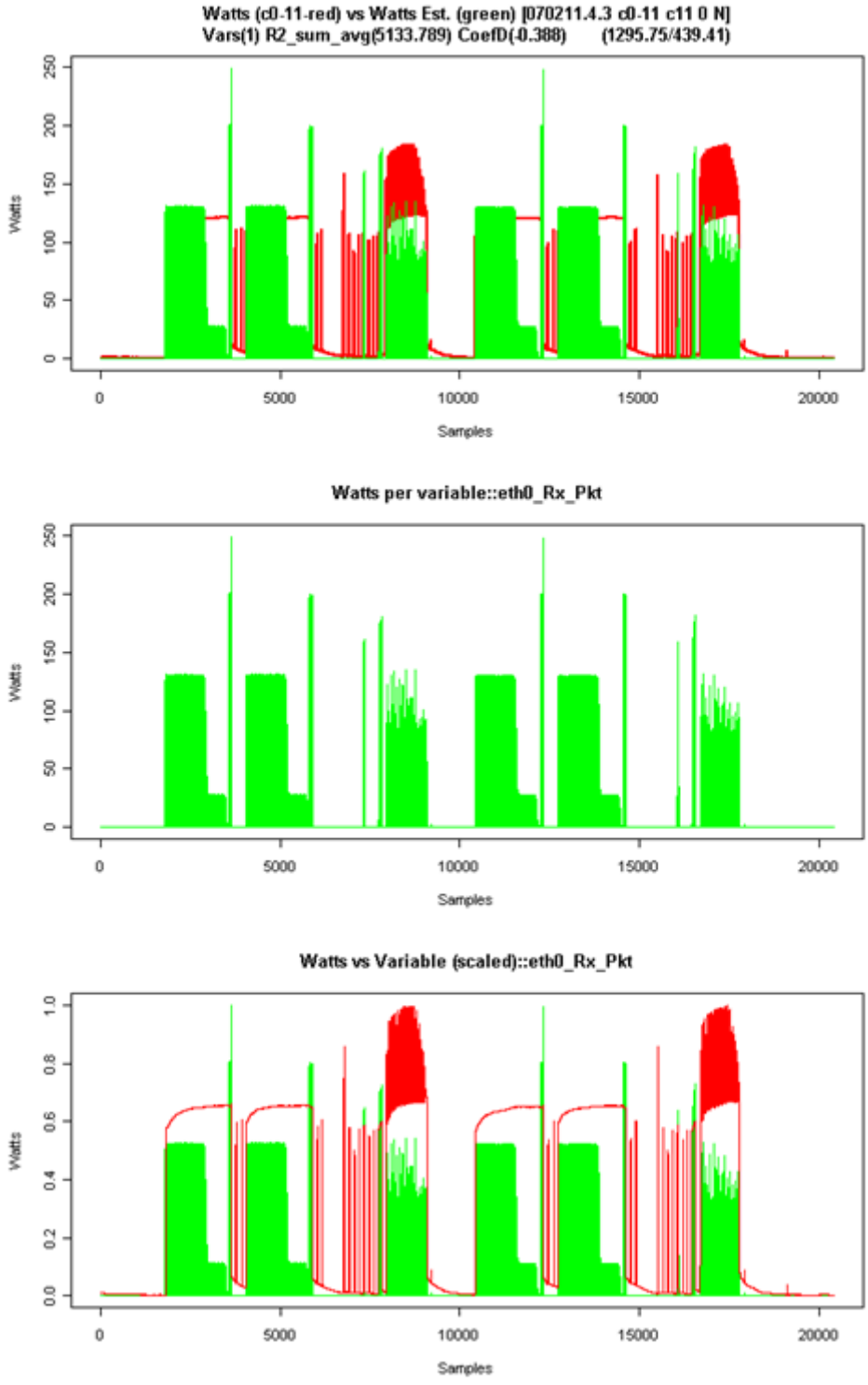


Figure 73 – /proc/net/dev.*eth0_**R**x_Pkt* single variable model (*n*20*p*40*c*160*aX*, Split HPCC).

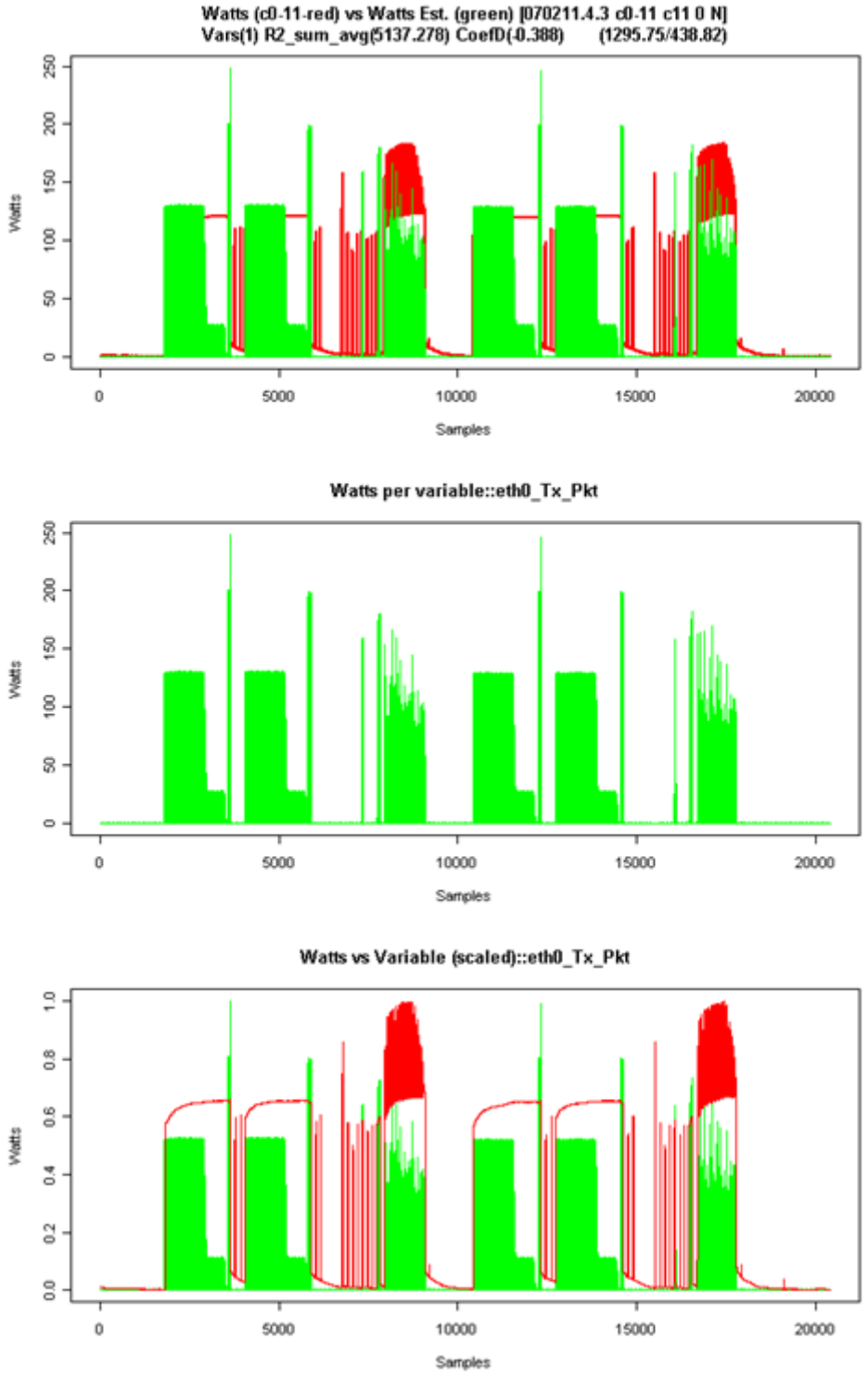


Figure 74 – /proc/net/dev.*eth0_**T**x_Pkt* single variable model (*n*20*p*40*c*160*aX*, Split HPCC).

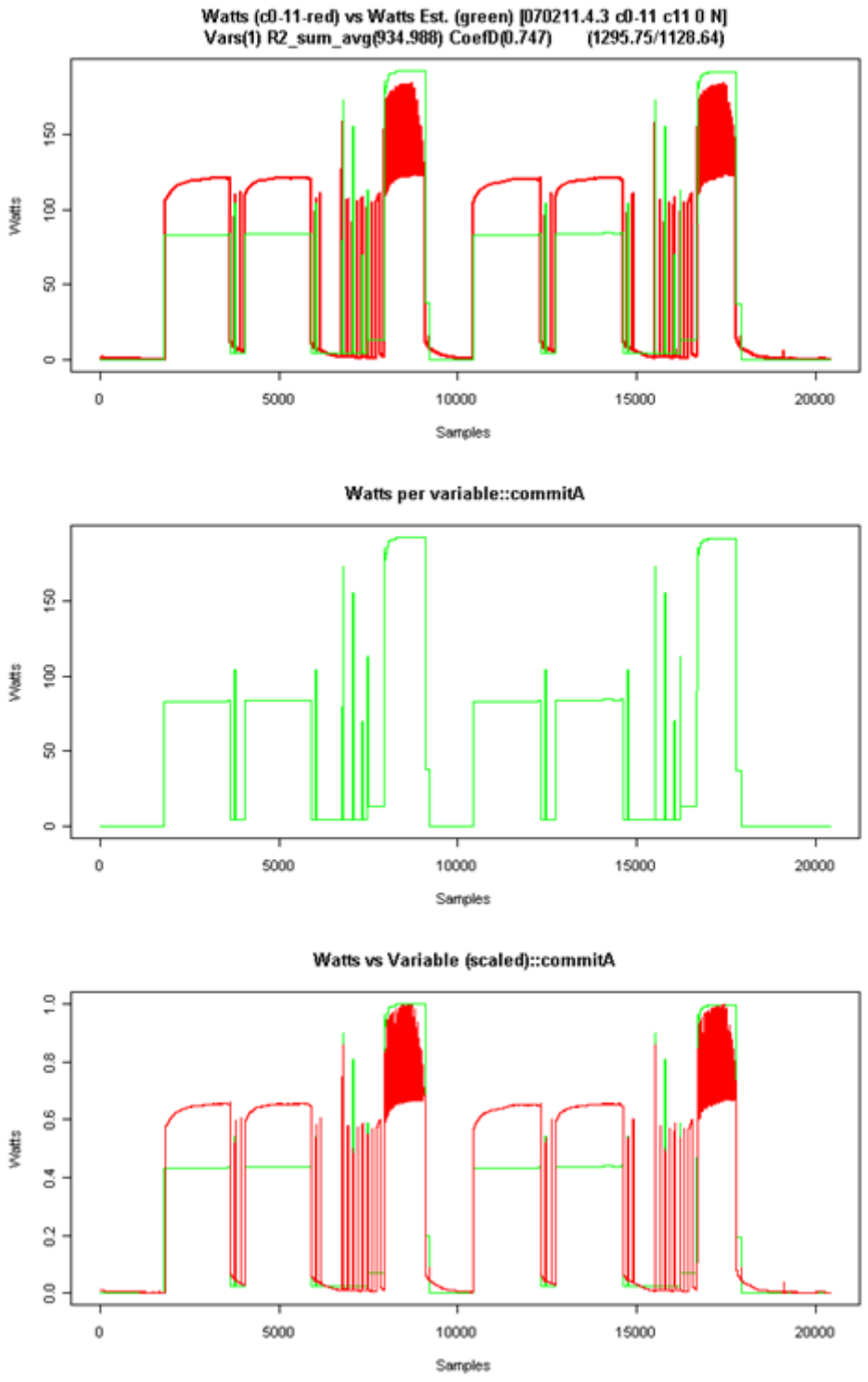


Figure 75 – /proc/meminfo.*commitA* single variable model (*n*20*p*40*c*160*aX*, Split HPCC).

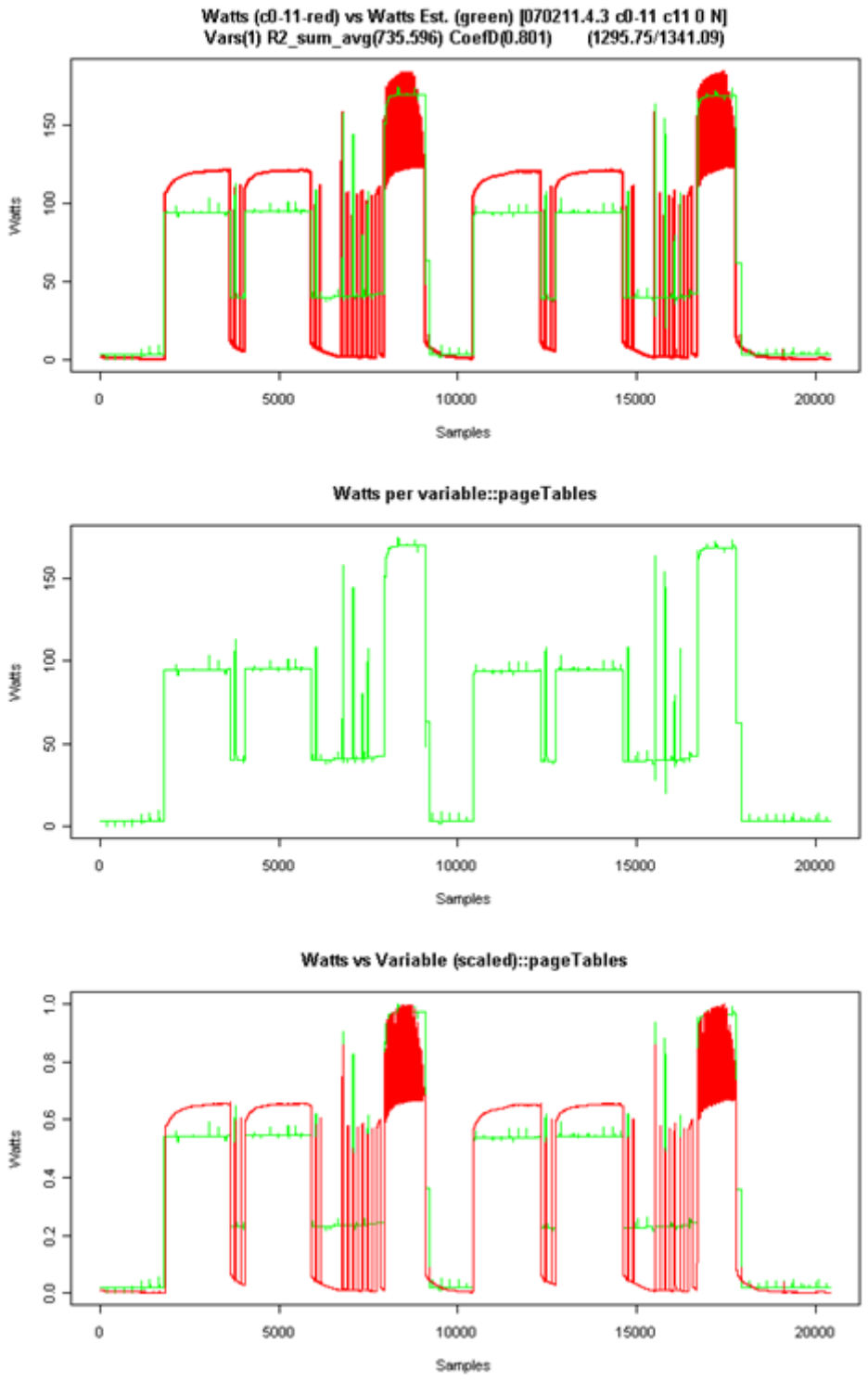


Figure 76 – /proc/meminfo.*pageTables* single variable model (*n*20*p*40*c*160*aX*, Split HPCC).

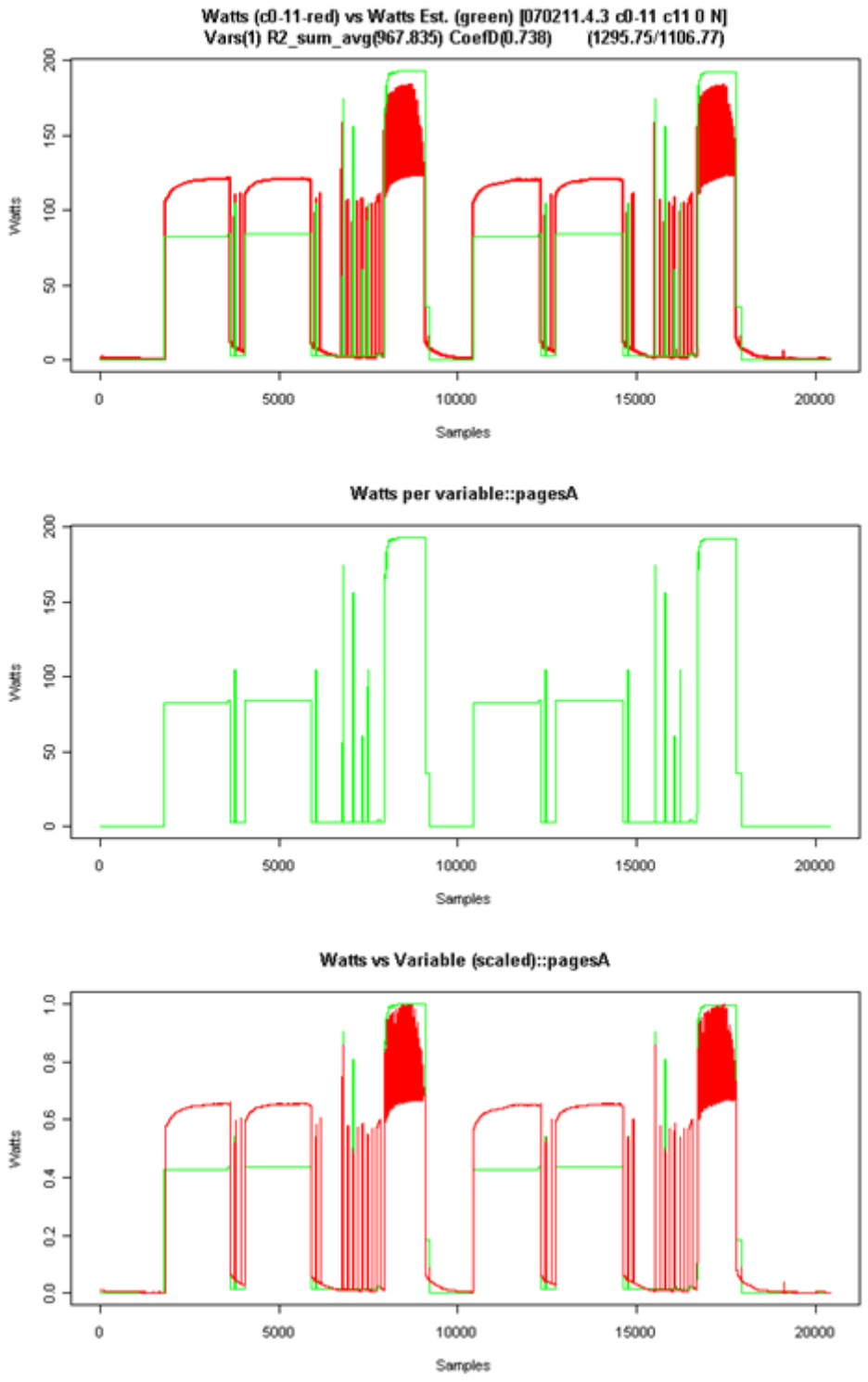


Figure 77 – /proc/meminfo.*pagesA* single variable model (*n*20*p*40*c*160*aX*, Split HPCC).

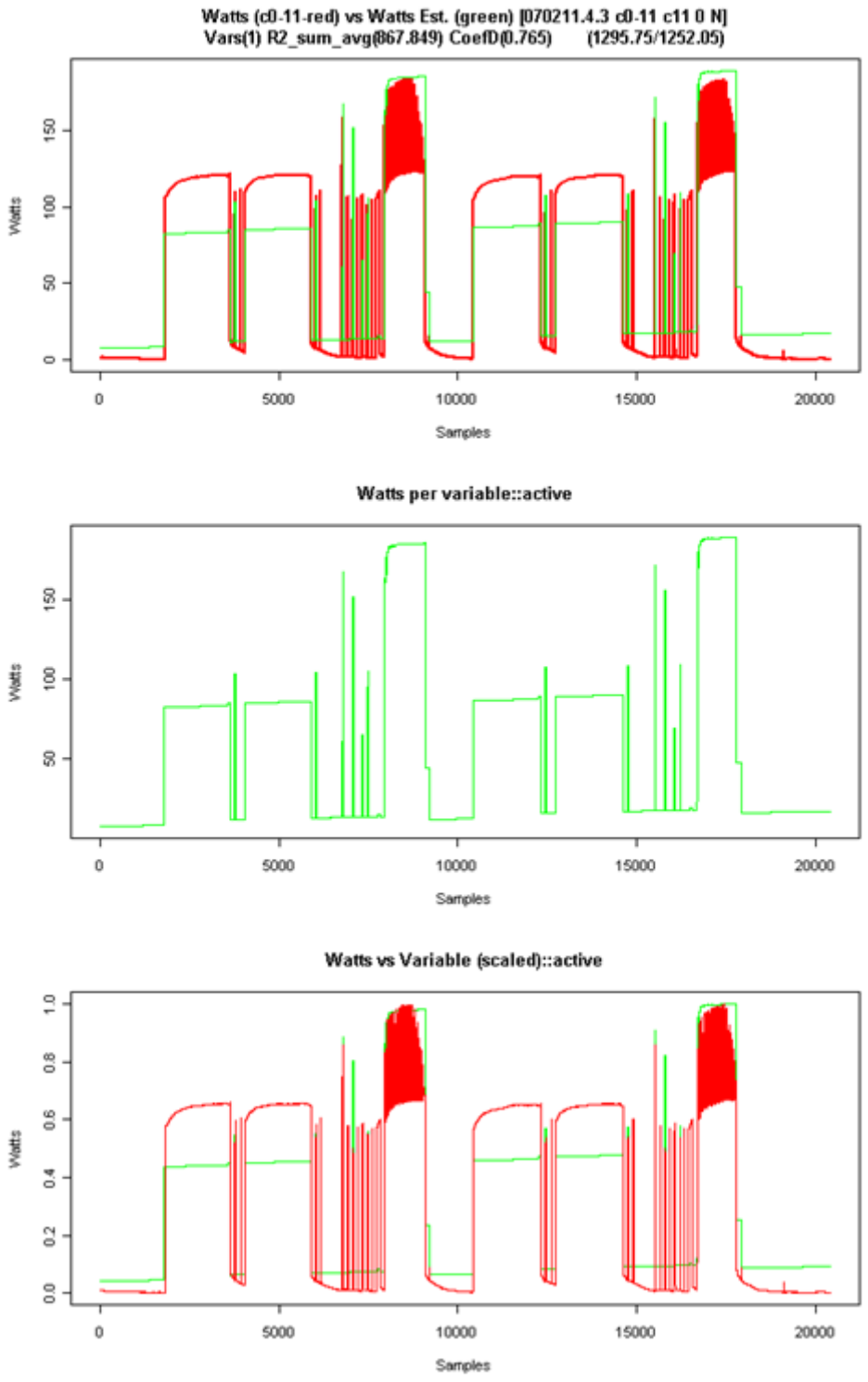


Figure 78 – /proc/meminfo.*active* single variable model (*n*20*p*40*c*160*aX*, Split HPCC).

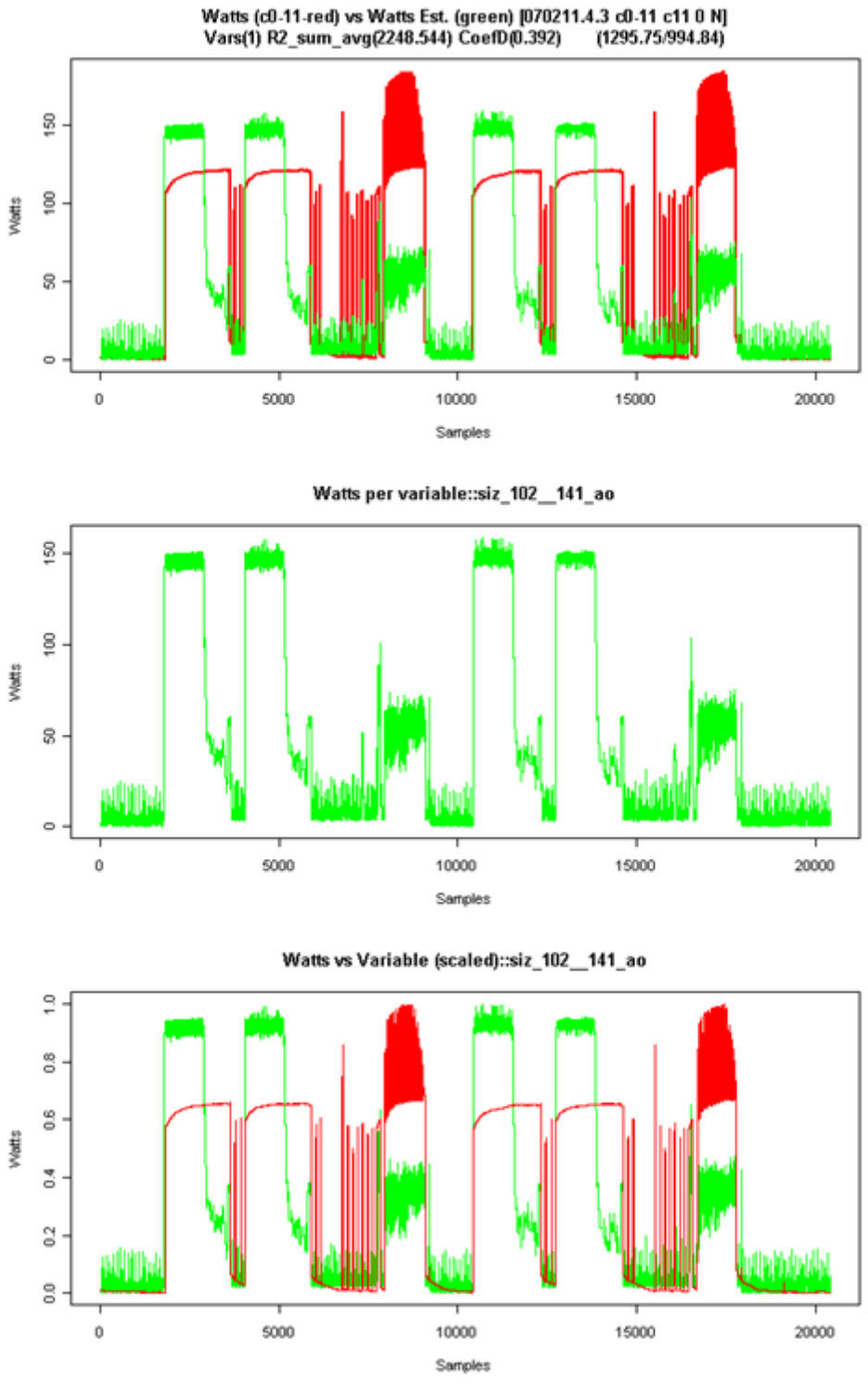


Figure 79 – /proc/slabinfo.*siz_102__141_ao* single variable model (*n*20*p*40*c*160*aX*).

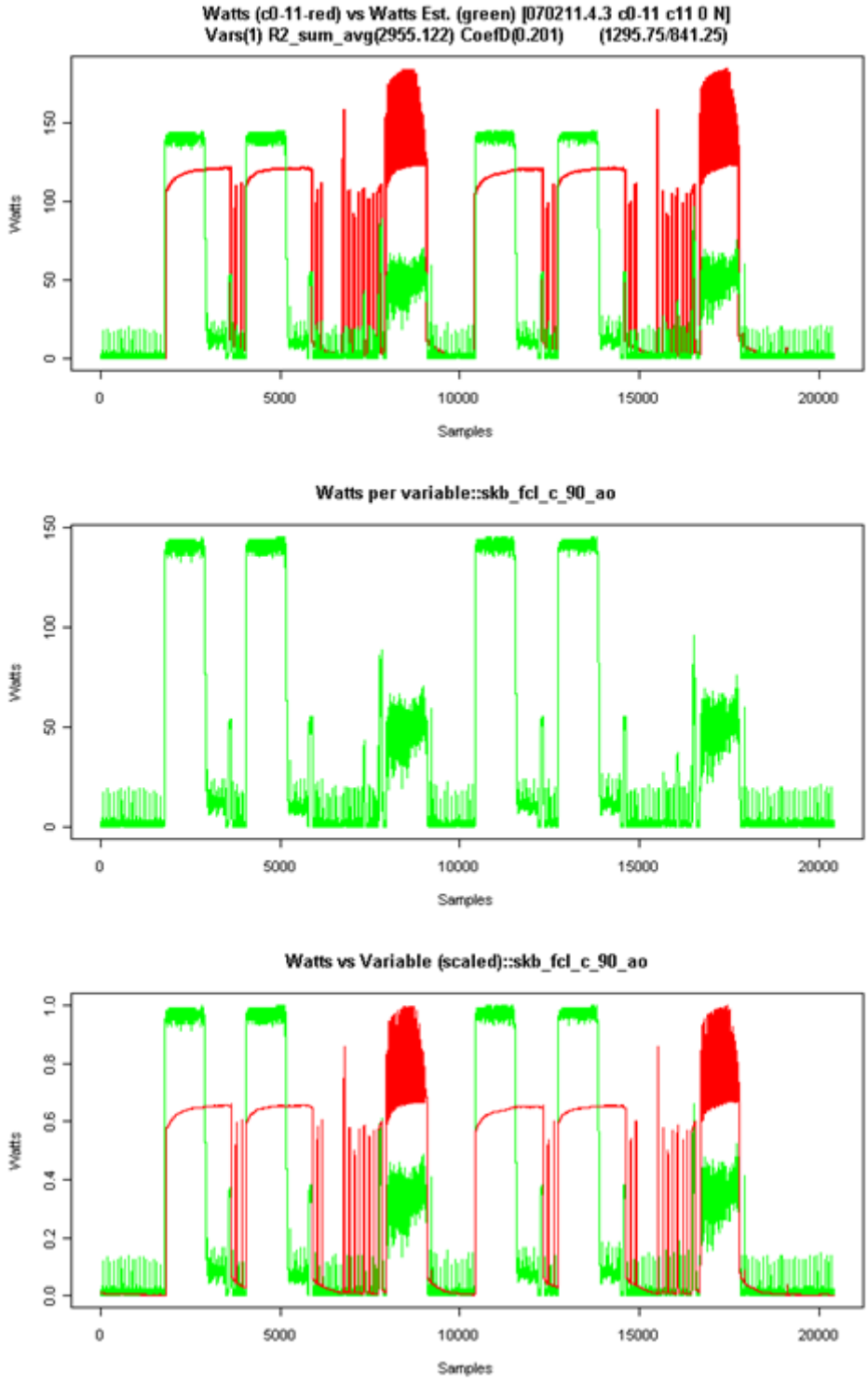


Figure 80 – /proc/slabinfo.*skb_fcl_c_90_ao* single variable model (*n*20*p*40*c*160*aX*).

Table 53 – Clusters identified for *n*20*p*40*c*160*aX*, Split HPCC.

Time processor spent executing in user mode

- /proc/stat.*user*
- /proc/slabinfo.*siz_204__139_sa*

Time processor spent executing in system mode

- /proc/stat.*sys*
- /proc/meminfo.*pageTables*
- /proc/meminfo.*commitA*
- /proc/interrupts.*nmi_CPUX* (where *X* is the core computing the benchmark)
- /proc/schedstat.*cpuX_9* (where *X* is the core computing the benchmark)

Time processor spent idling

- /proc/stat.*sysi*
- /proc/schedstat.*cpu0_6*
- /proc/buddyinfo.*normal_X* (where *X* is 1 through 4)

Ethernet interface interrupt and processor activity

- /proc/stat.*int1*
- /proc/stat.*int100*
- /proc/stat.*hirq*
- /proc/stat.*sirq*
- /proc/interrupts.*eth0_CPU7*
- /proc/slabinfo.*siz_102__141_ao*

Ethernet interface

- /proc/net/dev.eth0_{Rx,Tx}_{B,Pkt} (note has a similar envelope to Ethernet interrupts)

Table 53 - Clusters identified for *n*20*p*40*c*160*aX*, Split HPCC (Continued).

Disk utilization (sda)

- /proc/stat.*int76*
- /proc/interrupts.*usb_4_CPU1*
- /proc/diskstats.*sda_17_com_w*
- /proc/diskstats.*sda_17_sec_w*
- /proc/diskstats.*sda1_18_sec_r*
- /proc/diskstats.*sda_17_msec_w*
- /proc/diskstats.*sda_17_msec_io_w*

Memory (page utilization)

- /proc/meminfo.*active*
- /proc/meminfo.*pagesA*
- /proc/meminfo.*pageTables*
- /proc/meminfo.*commitA*
- /proc/meminfo.*mapped*
- /proc/vmstat.*nr_anon_pages*
- /proc/vmstat.*nr_page_table_pages*
- /proc/vmstat.*nr_mapped*

Memory (Page faults)

- /proc/vmstat.*numa_hit*
- /proc/vmstat.*numa_local*
- /proc/vmstat.*pgfault*
- /proc/vmstat.*pgalloc_normal*

Memory (Cache)

- /proc/meminfo.*cache*
- /proc/meminfo.*buf*
- /proc/vmstat.*nr_file_pages*

Table 54 – Least Squares results for *Algorithm 2* (*n*1*p*1*c*1*aX*-*t*1, c0-11).

Variable	x	Est. kWs	R2_sum_avg	CoefD	R2_sum_avg_x0	Corr_X0	CoefD_X0		Factor1	Factor2
user	0.364451	692.50	20.2657	0.9417	20.2657	**0.9705**	**0.9417**		**1.00**	-0.02
numa_hit	0.000071	2.95	18.9497	0.9455	938.5316	0.0476	-1.6985		0.01	**0.99**
inactive	0.000002	3.69	18.0404	0.9481	661.0636	0.2147	-0.9007		0.17	0.00
active	0.000001	79.38	17.7533	0.9490	52.1872	**0.9236**	**0.8499**		**0.94**	-0.02
normal_2	0.000101	12.23	17.5093	0.9497	945.8357	-0.8673	-1.7195		**-0.93**	-0.04
cpu7_9	0.002892	7.30	17.2210	0.9505	794.5286	0.3066	-1.2845		0.28	-0.02
cpu1_9	0.004692	1.33	17.0603	0.9509	929.4563	0.0976	-1.6724		0.08	-0.01
pgfree	0.000010	0.43	16.9127	0.9514	945.7633	0.0053	-1.7193		-0.02	0.00
cpu5_9	0.001654	3.21	16.7873	0.9517	839.2275	0.2412	-1.4130		0.23	-0.02
eth0_Rx_B	0.000004	0.15	16.7856	0.9517	938.4145	0.0111	-1.6982		0.00	0.16
fil___108_no	0.003688	0.18	16.7840	0.9517	924.3525	0.0697	-1.6578		0.07	0.00
normal_0	0.000002	0.06	16.7840	0.9517	946.0184	-0.1994	-1.7201		-0.22	-0.01
pgalloc_normal	0.000001	0.02	16.7839	0.9517	941.0605	0.0365	-1.7058		0.00	**0.93**
cpu1_4	0.000351	0.00	16.7839	0.9517	946.1186	0.0120	-1.7204		0.01	0.00
Watts									**0.97**	0.04

Table 55 – Least Squares results for *Algorithm 2* (*n*1*pXc*4*aX*-*t*1, c0-11).

Variable	x	Est. kWs	R2_sum_avg	CoefD	R2_sum_avg_x0	Corr_X0	CoefD_X0		Factor1	Factor2
user	0.100825	**328.54**	99.1429	0.9340	99.1429	**0.9682**	**0.9340**		**0.91**	0.36
nr_page_table_pages	0.003499	**119.03**	79.3527	0.9472	142.6997	**0.9513**	**0.9050**		**0.99**	0.16
sys	0.001580	1.26	65.9663	0.9561	1619.0831	0.4029	-0.0780		0.22	0.49
cpu1_9	0.019739	146.18	60.4991	0.9597	620.1869	0.7990	0.5871		**0.66**	0.48
cpu4_9	0.010357	50.49	54.1395	0.9640	1212.5679	0.5991	0.1927		0.52	0.30
cpu2_9	0.007210	48.27	51.3151	0.9658	797.1121	0.7404	0.4693		**0.65**	0.36
normal_5	0.000204	41.59	50.4663	0.9664	2214.7112	-0.8859	-0.4746		**-0.90**	-0.21
pagesA	0.000002	**127.31**	50.2495	0.9665	168.2239	**0.9478**	**0.8880**		**0.99**	0.15
nr_dirty	0.008804	10.50	50.0562	0.9667	1182.3347	0.5590	0.2128		**0.61**	0.01
pgalloc_dma32	0.000026	0.55	49.9662	0.9667	2200.7232	0.1079	-0.4653		0.14	-0.04
pgfree	0.000005	0.37	49.9241	0.9668	2223.3565	0.0484	-0.4803		0.01	0.10
pgalloc_normal	0.000007	0.38	49.8943	0.9668	2219.1738	0.0557	-0.4776		0.03	0.07
pgmajfault	0.288613	0.09	49.8753	0.9668	2234.3849	0.0134	-0.4877		0.01	0.02
cpu5_4	0.043549	1.80	49.8688	0.9668	1308.7819	0.5729	0.1286		**0.52**	0.25
sda_17_mer_w	0.005908	0.25	49.8639	0.9668	2179.7706	0.1155	-0.4513		0.13	0.00
cpu0_4	0.001029	0.07	49.8605	0.9668	2227.3919	0.0487	-0.4830		0.05	0.01
fil___108_no	0.001245	0.61	49.8588	0.9668	1835.4938	0.2667	-0.2221		0.23	0.14
lo_Rx_B	0.000038	0.06	49.8584	0.9668	2231.0198	-0.0006	-0.4855		-0.01	0.01
Watts									**0.90**	0.43

Table 56 – Least Squares results for *Algorithm 2* (*n*1*p*2*c*8*aX*-*t*1, c0-11).

Variable	x	Est, kWs	R2_sum_avg	CoefD	R2_sum_avg_x0	Corr_X0	CoefD_X0		Factor1	Factor2
cpu6_9	0.000029	0.08	125.2441	0.9202	125.2441	**0.9596**	**0.9202**		**0.96**	-0.10
inactive	0.000053	56.85	37.5299	0.9761	792.5195	0.7229	0.4951		0.57	-0.02
nr_anon_pages	0.000015	53.90	33.6575	0.9786	129.5574	**0.9592**	**0.9175**		**0.94**	-0.08
cpu5_4	3.226087	84.38	32.4739	0.9793	130.5629	**0.9581**	**0.9168**		**0.96**	-0.10
nr_page_table_page	0.000035	0.42	31.8765	0.9797	149.7591	**0.9543**	**0.9046**		**0.92**	0.00
cpu7_9	0.023525	62.41	31.5628	0.9799	128.5716	**0.9587**	**0.9181**		**0.96**	-0.10
sys	0.010246	4.33	31.3214	0.9800	1246.5255	0.5159	0.2058		0.45	-0.07
cpu4_9	0.021918	58.34	31.2440	0.9801	127.6081	**0.9589**	**0.9187**		**0.96**	-0.10
normal_0	0.000059	3.09	31.1818	0.9801	1775.8817	-0.1336	-0.1315		-0.06	**0.99**
fil___108_no	0.004215	4.01	31.1247	0.9802	1470.8654	0.3142	0.0629		0.25	0.17
pgfree	0.000008	0.30	31.0879	0.9802	1751.7843	0.1206	-0.1161		0.07	0.08
nmi_CPU4	1.989378	10.75	31.0663	0.9802	145.9828	**0.9528**	**0.9070**		**0.95**	-0.10
pageTables	0.000354	17.11	31.0559	0.9802	157.4114	**0.9516**	**0.8997**		**0.92**	-0.01
normal_4	0.000011	1.50	31.0536	0.9802	1777.7852	-0.7952	-0.1327		-0.85	0.31
normal_1	0.000003	0.30	31.0535	0.9802	1778.8738	-0.2763	-0.1334		-0.26	0.71
vm_are_s_116_no	0.000041	0.23	31.0535	0.9802	1102.1752	0.5457	0.2978		0.47	0.26
nfs	0.000773	0.00	31.0535	0.9802	1781.6962	0.0148	-0.1352		0.01	0.00
Watts									**0.94**	-0.07

Table 57 – Least Squares results for *Algorithm 2* (*n*1*p*2*c*8*aX*-*t*2, c0-11).

Variable	x	Est, kWs	R2_sum_avg	CoefD	R2_sum_avg_x0	Corr_X0	CoefD_X0		Factor1	Factor2
cpu6_9	0.005474	14.63	194.5854	0.9115	194.5854	**0.9552**	**0.9115**		-	-
inactive	0.000027	36.39	77.8017	0.9646	1322.9742	0.6850	0.3982		-	-
nr_anon_pages	0.000022	81.25	63.0242	0.9713	200.7870	**0.9555**	**0.9087**		-	-
nmi_CPU1	5.509922	29.33	58.0699	0.9736	201.5630	**0.9536**	**0.9083**		-	-
normal_5	0.000245	17.96	56.0229	0.9745	2647.1615	-0.8073	-0.2042		-	-
nmi_CPU3	5.067570	26.99	54.8988	0.9750	202.2990	**0.9534**	**0.9080**		-	-
nmi_CPU7	4.577181	24.37	54.4638	0.9752	204.6440	**0.9529**	**0.9069**		-	-
nmi_CPU2	4.309699	22.98	54.1705	0.9754	202.9302	**0.9533**	**0.9077**		-	-
nmi_CPU0	3.846144	20.63	54.0036	0.9754	210.5750	**0.9514**	**0.9042**		-	-
sys	0.005417	2.24	53.8919	0.9755	1862.2987	0.4982	0.1528		-	-
nmi_CPU5	2.125875	11.30	53.8326	0.9755	209.1368	**0.9519**	**0.9049**		-	-
nmi_CPU6	2.236584	12.02	53.7829	0.9755	215.8342	**0.9502**	**0.9018**		-	-
vm_are_s_116_no	0.000533	4.10	53.7446	0.9756	1669.0163	0.4918	0.2407		-	-
cpu1_9	0.005194	13.71	53.7130	0.9756	197.3145	**0.9547**	**0.9102**		-	-
nmi_CPU4	2.305032	12.31	53.6836	0.9756	204.8295	**0.9528**	**0.9068**		-	-
siz_204__139_sa	0.183297	0.07	53.6668	0.9756	2661.3546	0.0005	-0.2107		-	-
eth0_Rx_B	0.000001	0.05	53.6602	0.9756	2653.8204	0.0420	-0.2073		-	-
cpu3_9	0.005172	13.66	53.6567	0.9756	196.0050	**0.9550**	**0.9108**		-	-
cpu2_9	0.004516	11.94	53.6543	0.9756	195.5110	**0.9551**	**0.9111**		-	-
lo_Rx_B	0.000033	0.03	53.6523	0.9756	2661.5569	0.0006	-0.2108		-	-
nr_dirty	0.000220	0.17	53.6520	0.9756	1373.1008	0.6391	0.3753		-	-
procs	0.007347	0.01	53.6518	0.9756	2662.0116	-0.0055	-0.2110		-	-
cpu5_4	0.000235	0.01	53.6518	0.9756	2610.0914	0.1243	-0.1874		-	-
nr_unstable	0.004634	0.00	53.6517	0.9756	2662.2928	-0.0017	-0.2111		-	-

Table 58 – Least Squares results for *Algorithm 2* (*n*1*p*2*c*8*aX*-*t*3, c0-11).

Variable	x	Est, kWs	R2_sum_avg	CoefD	R2_sum_avg_x0	Corr_X0	CoefD_X0		Factor1	Factor2
nr_anon_pages	0.000023	82.13	245.8562	0.9047	245.8562	**0.9536**	**0.9047**		-	-
nmi_CPU2	5.860657	31.34	135.7908	0.9474	287.0748	**0.9433**	**0.8888**		-	-
inactive	0.000045	42.36	99.4013	0.9615	1489.6056	0.7101	0.4228		-	-
nmi_CPU1	4.852381	25.95	96.0799	0.9628	290.1820	**0.9426**	**0.8876**		-	-
normal_5	0.000230	13.54	93.8391	0.9636	3248.8717	-0.8380	-0.2590		-	-
cpu7_4	0.881621	23.16	92.2407	0.9643	374.0324	**0.9258**	**0.8551**		-	-
sys	0.013558	5.77	91.3007	0.9646	2296.3946	0.4802	0.1101		-	-
nmi_CPU3	5.179228	27.70	90.6195	0.9649	289.1951	**0.9428**	**0.8879**		-	-
nmi_CPU5	4.566268	24.42	90.2916	0.9650	291.3167	**0.9424**	**0.8871**		-	-
sda_17_num_io	0.196728	0.00	90.0674	0.9651	3267.3067	0.0013	-0.2661		-	-
nmi_CPU0	3.481781	18.73	89.9107	0.9652	295.5446	**0.9415**	**0.8855**		-	-
uni___40_no	0.018087	2.88	89.7807	0.9652	2659.5834	0.2516	-0.0306		-	-
nmi_CPU6	3.293236	17.72	89.6574	0.9653	299.0666	**0.9408**	**0.8841**		-	-
eth0_CPU6	0.000857	0.07	89.5742	0.9653	3263.8264	0.0028	-0.2647		-	-
nmi_CPU7	3.048031	16.29	89.5172	0.9653	293.1641	**0.9420**	**0.8864**		-	-
nmi_CPU4	2.498035	13.39	89.4822	0.9653	289.0738	**0.9428**	**0.8880**		-	-
sda1_18_msec_r	0.000624	0.17	89.4633	0.9653	3171.8293	0.1441	-0.2291		-	-
fil___108_no	0.001311	1.05	89.4509	0.9653	2711.1163	0.2183	-0.0506		-	-
lo_Rx_B	0.000081	0.07	89.4459	0.9653	3266.0234	0.0013	-0.2656		-	-
ctxt	0.000142	0.20	89.4433	0.9653	3253.0856	0.0006	-0.2606		-	-

Table 59 – Least Squares results for *Algorithm 2* (*n*2*p*4*c*16*aX*-*t*1, c0-11).

Variable	x	Est, kWs	R2_sum_avg	CoefD	R2_sum_avg_x0	Corr_X0	CoefD_X0		Factor1	Factor2
cpu6_9	0.049627	218.32	177.8120	0.9393	177.8120	**0.9694**	**0.9393**		**0.93**	0.37
commitA	0.000001	24.80	71.2969	0.9757	293.5509	**0.9516**	**0.8999**		0.66	0.74
dirty	0.008263	34.23	64.6894	0.9779	1460.0675	0.7359	0.5019		0.54	0.48
user	0.019119	39.70	62.2688	0.9788	515.8437	**0.9155**	**0.8240**		0.68	0.64
bio___86_ao	0.015092	5.92	60.2535	0.9794	3002.8666	0.3761	-0.0245		0.20	0.36
pagesA	0.000006	130.29	59.3962	0.9797	285.6692	**0.9537**	**0.9025**		0.67	0.74
normal_4	0.000094	13.23	58.4128	0.9801	3994.0610	-0.9062	-0.3626		-0.77	-0.54
siz_512__143_no	0.006421	3.16	58.2795	0.9801	2515.6543	0.5016	0.1417		0.40	0.27
sgp_16__30_ao	0.173746	0.96	58.2326	0.9801	3568.7959	0.2610	-0.2176		0.19	0.18
ide0_CPU1	0.036639	1.23	58.1988	0.9801	2750.4822	0.4811	0.0616		0.39	0.27
cpu3_9	0.007422	32.46	58.1777	0.9802	181.1944	**0.9690**	**0.9382**		**0.93**	0.37
siz_128__150_ao	0.000822	2.90	58.1614	0.9802	2051.9928	0.5478	0.2999		0.50	0.22
sgp_32__29_ao	0.198780	0.50	58.1490	0.9802	3718.6109	0.1882	-0.2687		0.09	0.19
jou_hea__36_no	0.001950	2.19	58.1416	0.9802	2284.2046	0.4951	0.2207		0.28	0.46
cpu5_9	0.005646	24.69	58.1364	0.9802	180.0461	**0.9692**	**0.9386**		**0.93**	0.37
cpu1_9	0.003904	17.07	58.1338	0.9802	181.1651	**0.9690**	**0.9382**		**0.93**	0.37
sda1_18_msec_r	0.000152	0.06	58.1320	0.9802	3883.4035	0.1456	-0.3249		0.08	0.14
usb_4_CPU4	0.003644	0.05	58.1316	0.9802	3882.1271	0.1120	-0.3244		0.05	0.13
cpu7_9	0.001594	6.97	58.1313	0.9802	180.7767	**0.9690**	**0.9383**		**0.93**	0.37
sda_17_msec_io	0.002136	0.02	58.1312	0.9802	3955.2521	0.0835	-0.3494		0.03	0.10
Watts									**0.83**	0.54

Table 60 – Least Squares results for *Algorithm 2* (*n*2*p*4*c*16*aX*-*t*1, c0-12).

Variable	x	Est, kWs	R2_sum_avg	CoefD	R2_sum_avg_x0	Corr_X0	CoefD_X0		Factor1	Factor2
nmi_CPU2	32.676404	287.00	175.5019	0.9377	175.5019	**0.9692**	**0.9377**		**0.96**	0.13
active	0.000004	94.60	72.1681	0.9744	286.1422	**0.9515**	**0.8984**		**0.94**	0.19
nr_dirty	0.006324	6.57	66.1812	0.9765	1565.6630	0.7112	0.4440		0.69	0.18
normal_4	0.000145	19.64	62.4122	0.9778	3979.9138	-0.9044	-0.4133		-0.92	-0.12
user	0.020262	42.38	59.9958	0.9787	513.6637	**0.9149**	**0.8176**		**0.91**	0.15
nr_anon_pages	0.000010	54.25	59.3585	0.9789	332.5201	**0.9451**	**0.8819**		**0.94**	0.19
bio_1__85_ao	0.007772	3.49	58.7801	0.9791	3044.7470	0.3344	-0.0812		0.18	**0.94**
dirty	0.004682	19.44	58.3793	0.9793	1572.4836	0.7097	0.4416		0.70	0.11
mapped	0.000125	6.65	58.2450	0.9793	1876.2157	0.5831	0.3338		0.56	0.16
sgp_32__29_ao	0.555624	0.88	58.1810	0.9793	3628.9478	0.2519	-0.2886		0.14	0.67
cpu3_4	0.157307	6.85	58.1383	0.9794	360.5564	**0.9366**	**0.8720**		**0.94**	0.12
ide0_CPU1	0.032115	1.04	58.1046	0.9794	2804.4283	0.4610	0.0042		0.45	0.15
pgalloc_dma32	0.000015	0.28	58.0824	0.9794	3829.0812	0.1841	-0.3597		0.18	0.06
siz_102__141_no	0.002964	2.84	58.0632	0.9794	1545.0324	0.6896	0.4514		0.66	0.22
sda_17_num_io	0.089229	0.00	58.0454	0.9794	4006.4125	0.0029	-0.4227		0.00	0.00
bio___86_ao	0.005601	1.98	58.0358	0.9794	3065.5723	0.3620	-0.0886		0.20	**0.95**
fil___108_no	0.001158	2.09	58.0263	0.9794	2349.6222	0.4335	0.1657		0.42	0.07
cpu7_4	0.029774	1.31	58.0184	0.9794	1118.2979	**0.8024**	**0.6029**		**0.80**	0.10
Watts									**0.98**	0.17

Table 61 – Least Squares results for *Algorithm 2* (*n*4*p*8*c*32*aX*-*t*1, c0-11).

Variable	x	Est, kWs	R2_sum_avg	CoefD	R2_sum_avg_x0	Corr_X0	CoefD_X0		Factor1	Factor2
cpu1_4	4.247929	286.14	374.1401	0.9091	374.1401	**0.9555**	**0.9091**		**0.96**	-
pageTables	0.000224	22.97	195.1555	0.9526	542.5663	**0.9338**	**0.8682**		**0.95**	-
nr_dirty	0.028031	39.19	181.7087	0.9558	2876.1079	0.6794	0.3011		0.67	-
commitA	0.000006	225.15	173.5545	0.9578	667.4671	**0.9284**	**0.8378**		**0.95**	-
normal_7	0.001578	27.61	170.3062	0.9586	7989.1219	-0.9169	-0.9414		-0.95	-
bio___86_ao	0.012978	6.45	167.8731	0.9592	5458.4701	0.3873	-0.3264		0.39	-
user	0.055101	135.05	167.1619	0.9594	2305.9703	0.7720	0.4396		0.77	-
sys	0.046863	129.58	165.4255	0.9598	3224.3550	0.6330	0.2165		0.64	-
siz_204__139_sa	0.054394	8.49	164.5633	0.9600	3219.7114	0.6609	0.2176		0.67	-
jou_hea__36_no	0.010560	8.85	164.2411	0.9601	4405.2339	0.4748	-0.0705		0.48	-
siz_204__139_ao	0.015564	18.94	163.9351	0.9602	1752.2151	0.7726	0.5742		0.79	-
ide0_CPU1	0.056216	2.78	163.8163	0.9602	5386.0218	0.4217	-0.3088		0.43	-
pgfree	0.000007	0.65	163.7326	0.9602	7730.1055	0.1415	-0.8784		0.13	-
usb_4_CPU4	0.043047	0.51	163.6834	0.9602	7896.9838	0.0599	-0.9190		0.06	-
ide0_CPU5	0.620283	0.06	163.6417	0.9602	8062.6643	0.0129	-0.9592		0.01	-
cpu5_6	0.002226	0.15	163.6391	0.9602	8063.9032	-0.1611	-0.9595		-0.17	-
cpu4_4	0.000685	0.05	163.6391	0.9602	825.7953	**0.9034**	**0.7993**		**0.92**	-
Watts									**0.98**	-

Table 62 – Least Squares results for *Algorithm 2* (*n*4*p*8*c*32*aX*-*t*1, c0-12).

Variable	x	Est, kWs	R2_sum_avg	CoefD	R2_sum_avg_x0	Corr_X0	CoefD_X0		Factor1	Factor2
cpu2_9	0.051645	349.64	295.0032	0.9173	295.0032	**0.9578**	**0.9173**		**0.93**	0.36
commitA	0.000000	1.19	156.1097	0.9562	506.2856	**0.9312**	**0.8581**		0.64	0.77
nr_dirty	0.037533	49.40	144.8945	0.9594	2119.2147	0.7140	0.4058		0.45	0.58
jou_hea__36_no	0.016879	12.83	143.0585	0.9599	3529.7332	0.4906	0.0104		0.26	0.48
user	0.011250	27.68	141.2625	0.9604	1687.4255	0.7862	0.5269		0.49	0.70
active	0.000001	27.84	139.6882	0.9608	461.4194	**0.9368**	**0.8706**		0.65	0.76
siz_204__139_sa	0.042207	6.99	139.4434	0.9609	2335.8552	0.6877	0.3451		0.64	0.28
bio___86_ao	0.002006	1.03	139.3808	0.9609	4148.1041	0.4118	-0.1630		0.26	0.35
cpu5_4	0.109466	7.40	139.3305	0.9609	769.7856	**0.8906**	**0.7842**		**0.86**	0.34
usb_4_CPU4	0.049023	0.60	139.2561	0.9610	5972.7287	0.0672	-0.6746		-0.01	0.13
cpu1_6	0.009338	0.33	139.2142	0.9610	6088.5332	-0.0699	-0.7070		-0.07	-0.03
cpu5_9	0.012165	82.34	139.1765	0.9610	296.9538	**0.9575**	**0.9167**		**0.93**	0.36
pagesA	0.000005	175.97	139.1349	0.9610	477.6596	**0.9357**	**0.8661**		0.65	0.76
cpu3_9	0.007798	52.79	139.1224	0.9610	297.3430	**0.9575**	**0.9166**		**0.93**	0.36
jou_hea__36_ao	0.002557	1.79	139.1167	0.9610	4203.5315	0.3664	-0.1785		0.20	0.35
uni___40_no	0.001331	0.40	139.1161	0.9610	3887.1813	0.2716	-0.0898		0.29	0.06
ide0_CPU1	0.003175	0.16	139.1157	0.9610	4044.3122	0.4447	-0.1339		0.36	0.25
sda1_18_sec_r	0.000601	0.04	139.1155	0.9610	5885.9700	0.1360	-0.6502		0.07	0.13
Watts									**0.82**	0.53

Table 63 – Least Squares results for *Algorithm 2* (*n*4*p*8*c*32*aX*-*t*2, c0-11).

Variable	x	Est, kWs	R2_sum_avg	CoefD	R2_sum_avg_x0	Corr_X0	CoefD_X0		Factor1	Factor2
cpu1_4	4.504080	303.41	347.4880	0.9149	347.4880	**0.9576**	**0.9149**		**0.99**	0.13
commitA	0.000001	30.00	180.9280	0.9557	614.4380	**0.9311**	**0.8495**		**0.80**	0.60
nr_dirty	0.030694	43.17	168.1377	0.9588	2673.1136	0.6882	0.3452		0.55	0.45
normal_5	0.000709	30.04	160.6929	0.9606	7424.7379	-0.8625	-0.8187		-0.85	-0.32
bio___86_ao	0.012383	6.27	158.4945	0.9612	5076.6537	0.3948	-0.2435		0.31	0.27
pagesA	0.000006	222.68	157.5553	0.9614	585.6106	**0.9358**	**0.8566**		**0.81**	0.59
cpu0_9	0.031918	216.08	156.4851	0.9617	349.8135	**0.9572**	**0.9143**		**0.99**	0.13
user	0.008557	20.97	155.8625	0.9618	2131.8925	0.7780	0.4778		0.64	0.57
siz_204__139_sa	0.068732	10.73	155.1196	0.9620	2985.8484	0.6704	0.2686		0.67	0.11
jou_hea__36_no	0.011146	9.63	154.8022	0.9621	4122.3555	0.4782	-0.0098		0.35	0.38
pgfree	0.000011	0.99	154.6231	0.9621	7167.6697	0.1457	-0.7557		0.15	-0.02
ide0_CPU5	0.778728	0.07	154.5630	0.9621	7475.7760	0.0136	-0.8312		0.03	-0.06
usb_4_CPU4	0.037635	0.47	154.5165	0.9622	7337.1619	0.0563	-0.7972		0.01	0.13
siz_204__139_ao	0.004096	5.00	154.4916	0.9622	1640.9324	**0.7834**	**0.5981**		**0.83**	0.04
ide0_CPU1	0.023442	1.16	154.4719	0.9622	4997.6759	0.4339	-0.2242		0.40	0.15
sda_17_msec_w	0.000665	0.04	154.4713	0.9622	7365.1320	0.0926	-0.8041		0.06	0.10
Watts									**0.92**	0.32

Table 64 – Least Squares results for *Algorithm 2* (*n*4*p*8*c*32*aX*-*t*2, c0-12).

Variable	x	Est, kWs	R2_sum_avg	CoefD	R2_sum_avg_x0	Corr_X0	CoefD_X0		Factor1	Factor2
cpu2_9	0.050820	344.04	304.4708	0.9154	304.4708	**0.9568**	**0.9154**		**0.95**	0.32
commitA	0.000000	3.47	161.1193	0.9552	522.5372	**0.9298**	**0.8548**		0.67	0.74
nr_dirty	0.037434	49.10	149.5398	0.9584	2186.2467	0.7105	0.3925		0.47	0.56
jou_hea__36_no	0.017173	12.94	147.6337	0.9590	3638.3564	0.4865	-0.0110		0.27	0.47
user	0.011202	27.56	145.7785	0.9595	1741.5931	0.7832	0.5160		0.52	0.68
active	0.000006	207.51	144.1429	0.9599	476.2207	**0.9356**	**0.8677**		0.68	0.73
sda1_18_sec_r	0.001684	0.11	144.0617	0.9600	6074.8932	0.1346	-0.6881		0.08	0.13
cpu5_4	0.108944	7.37	144.0074	0.9600	794.4948	**0.8882**	**0.7792**		**0.88**	0.30
siz_204__139_sa	0.044975	7.45	143.7714	0.9600	2410.8389	0.6830	0.3301		0.65	0.25
normal_5	0.000056	2.58	143.7055	0.9601	6262.7816	-0.8739	-0.7403		-0.78	-0.48
cpu1_6	0.008292	0.26	143.6663	0.9601	6283.9864	-0.0648	-0.7462		-0.07	-0.03
usb_4_CPU4	0.043463	0.52	143.6286	0.9601	6164.2155	0.0666	-0.7129		-0.01	0.13
cpu5_9	0.012149	82.24	143.5902	0.9601	306.4841	**0.9565**	**0.9148**		**0.95**	0.32
bio___86_ao	0.003094	1.58	143.5684	0.9601	4279.8180	0.4073	-0.1893		0.27	0.34
cpu3_9	0.008046	54.47	143.5549	0.9601	306.8855	**0.9565**	**0.9147**		**0.95**	0.32
jou_hea__36_ao	0.000777	0.54	143.5544	0.9601	4332.6080	0.3634	-0.2040		0.21	0.34
ide0_CPU1	0.000306	0.02	143.5544	0.9601	4174.1413	0.4387	-0.1599		0.37	0.23
Watts									**0.84**	0.49

Table 65 – Least Squares results for *Algorithm 2* (*n*20*p*40*c*160*aX*-*t*1, c0-11).

Variable	x	Est, kWs	R2_sum_avg	CoefD	R2_sum_avg_x0	Corr_X0	CoefD_X0		Factor1	Factor2
cpu6_9	0.000756	7.85	150.7704	0.9593	150.7704	**0.9799**	**0.9593**		**0.88**	0.47
active	0.000002	49.23	92.1116	0.9751	867.8492	**0.8765**	**0.7654**		0.46	**0.88**
siz_204__139_ao	0.117726	220.96	87.4299	0.9764	521.3942	**0.9276**	**0.8591**		0.69	0.63
user	0.029550	40.47	81.1006	0.9781	5130.5139	0.4800	-0.3866		0.10	0.66
eth0_CPU7	0.000641	82.73	76.4251	0.9793	2389.6781	**0.7091**	**0.3541**		**0.87**	0.01
commitA	0.000005	115.46	75.4997	0.9796	934.9878	0.8833	0.7473		0.48	**0.88**
cpu0_9	0.074196	769.50	75.3455	0.9796	151.2815	**0.9799**	**0.9591**		**0.88**	0.47
nfs	0.146884	0.07	75.3123	0.9796	7720.2780	0.0148	-1.0866		0.00	0.02
skb_hea_c_91_ao	0.001675	13.41	75.2879	0.9797	487.7388	**0.9323**	**0.8682**		**0.82**	0.46
pgmajfault	0.084063	0.01	75.2852	0.9797	7724.7852	-0.0065	-1.0878		0.00	-0.01
procs_r	0.016142	2.12	75.2844	0.9797	553.1749	**0.9307**	**0.8505**		**0.83**	0.45
usb_4_CPU1	0.003287	0.05	75.2842	0.9797	7286.6154	0.1681	-0.9694		0.13	0.11
cpu3_4	0.003103	0.32	75.2840	0.9797	2018.4032	0.7571	0.4545		0.68	0.36
sgp_16__30_ao	0.004513	0.14	75.2839	0.9797	4718.8346	0.4782	-0.2754		0.42	0.24
eth0_CPU6	0.000020	0.01	75.2838	0.9797	7710.5925	0.0181	-1.0840		0.03	-0.01
Watts									**0.81**	0.57

Table 66 – Least Squares results for *Algorithm 2* (*n*20*p*40*c*160*aX*-*t*1, c0-12).

Variable	x	Est, kWs	R2_sum_avg	CoefD	R2_sum_avg_x0	Corr_X0	CoefD_X0		Factor1	Factor2
cpu6_9	0.000042	0.44	165.2735	0.9503	165.2735	**0.9753**	**0.9503**		**0.87**	0.49
active	0.000000	12.97	140.3426	0.9578	878.9595	**0.8593**	**0.7356**		0.56	0.71
user	0.126603	171.60	133.1034	0.9600	4654.8634	0.4672	-0.4004		0.07	0.70
siz_204__139_ao	0.089512	167.64	128.8115	0.9612	614.3439	**0.9029**	**0.8152**		0.67	0.64
eth0_CPU7	0.000323	41.43	124.2821	0.9626	2084.8414	**0.7180**	**0.3728**		**0.98**	-0.18
sys	0.092700	607.73	123.0793	0.9630	1017.2659	**0.8529**	**0.6940**		**0.90**	0.25
uni___40_no	0.053423	13.11	122.3757	0.9632	4437.5326	0.2953	-0.3350		0.24	0.14
nr_anon_pages	0.000010	57.29	122.1693	0.9632	964.7695	**0.8668**	**0.7097**		0.57	0.71
pgmajfault	0.614382	0.09	122.0295	0.9633	6904.0742	0.0002	-1.0771		-0.01	0.00
int1	0.000502	64.86	121.9761	0.9633	2067.3250	**0.7192**	**0.3780**		**0.98**	-0.17
sgp_32__29_ao	0.288899	0.61	121.9351	0.9633	6488.3981	0.1716	-0.9520		0.10	0.15
ide0_CPU5	0.450610	0.04	121.9185	0.9633	6901.8419	0.0124	-1.0764		0.00	0.04
skb_fcl_c_90_ao	0.001141	5.18	121.9073	0.9633	2699.9901	**0.6344**	**0.1877**		**0.80**	-0.05
crq_poo__45_ao	0.004943	1.60	121.8973	0.9633	4755.9518	0.2146	-0.4308		0.19	0.10
cpu5_4	0.050294	5.22	121.8876	0.9633	632.1345	**0.9087**	**0.8098**		**0.81**	0.46
ide0_CPU1	0.011849	0.87	121.8796	0.9633	4569.6855	0.4179	-0.3748		0.37	0.19
nfs	0.041467	0.04	121.8723	0.9633	6898.2990	0.0219	-1.0753		0.00	0.04
cpu1_9	0.004427	45.89	121.8676	0.9633	168.6197	**0.9749**	**0.9493**		**0.87**	0.49
usb_4_CPU4	0.017122	0.10	121.8665	0.9633	6867.2111	-0.0603	-1.0660		-0.09	0.03
cpu5_9	0.002337	24.23	121.8654	0.9633	168.1951	**0.9749**	**0.9494**		**0.87**	0.49
int16	0.002721	0.50	121.8650	0.9633	5118.3852	-0.0003	-0.5399		0.00	0.00
sda4_21_mer_r	0.009773	0.05	121.8649	0.9633	6737.1579	-0.0059	-1.0269		-0.01	0.00
Watts									**0.82**	0.53

Table 67 – Least Squares results for *Algorithm 2* (*n*20*p*40*c*160*aX-t*1, c0-11).

Variable	x	Est, kWs	R2_sum_avg	CoefD	R2_sum_avg_x0	Corr_X0	CoefD_X0		Factor1	Factor2
cpu6_9	0.000008	0.08	129.0229	0.9651	129.0229	**0.9829**	**0.9651**		**0.92**	0.34
active	0.000000	1.18	74.5631	0.9799	848.3530	**0.8791**	**0.7708**		0.57	0.61
user	0.126594	173.39	69.9082	0.9811	5008.3452	0.4840	-0.3534		0.08	**0.99**
siz_204__139_ao	0.126113	236.96	64.0782	0.9827	504.4810	0.9300	0.8637		0.77	0.41
eth0_CPU7	0.000919	118.61	60.1614	0.9837	2320.6158	**0.7165**	**0.3729**		**0.88**	-0.12
sys	0.095826	626.85	58.3022	0.9842	1102.0219	**0.8577**	**0.7022**		**0.97**	-0.07
nr_anon_pages	0.000025	140.75	56.9641	0.9846	942.8141	**0.8857**	**0.7452**		0.58	0.61
pgmajfault	0.815577	0.12	56.6789	0.9847	7544.8925	0.0026	-1.0388		0.00	0.00
nfs	0.138905	0.07	56.6502	0.9847	7541.1402	0.0151	-1.0378		0.00	0.01
ide0_CPU1	0.014399	1.06	56.6425	0.9847	4947.3052	0.4323	-0.3369		0.35	0.19
usb_4_CPU1	0.013602	0.22	56.6394	0.9847	7117.1466	0.1705	-0.9232		0.15	0.04
lo_Rx_Pkt	0.025895	0.44	56.6371	0.9847	7057.0785	-0.0001	-0.9070		0.00	0.00
dirty	0.000119	0.73	56.6363	0.9847	3190.2701	0.6263	0.1379		0.31	0.53
dma32_0	0.000001	0.43	56.6363	0.9847	5158.5922	-0.4753	-0.3940		-0.03	-0.68
Watts									**0.86**	0.43

Table 68 – *Algorithm 2* percent error in energy estimates across all trials.

Configuration / Node / Experiment	Tot. kWs	Est. kWs	% Error
(*n1p1c1aX-t1w11*)	793.49	803.44	1.25
(*n1p1c4aX-t1w11*)	863.25	877.35	1.63
(*n1p2c8aX-t1w11*)	356.32	358.00	0.47
(*n1p2c8aX-t2w11*)	350.62	356.14	1.57
(*n1p2c8aX-t3w11*)	342.84	346.91	1.19
(*n2p4c16aX-t1w11*)	551.46	558.75	1.32
n2p4c16aX-t1w12	554.98	551.28	0.67
(*n4p8c32aX-t1w11*)	908.72	912.61	0.43
(*n4p8c32-t1w12*)	794.41	798.43	0.51
(*n4p8c32-t2w11*)	893.25	900.70	0.83
(*n4p8c32-t2w12*)	794.27	801.74	0.94
(*n20p30c160aX-t1w11*)	1295.75	1302.34	0.51
n20p40c160-t1w12	1222.86	1221.51	0.11
n20p40c160aX-t2w11	1301.19	1300.88	0.02

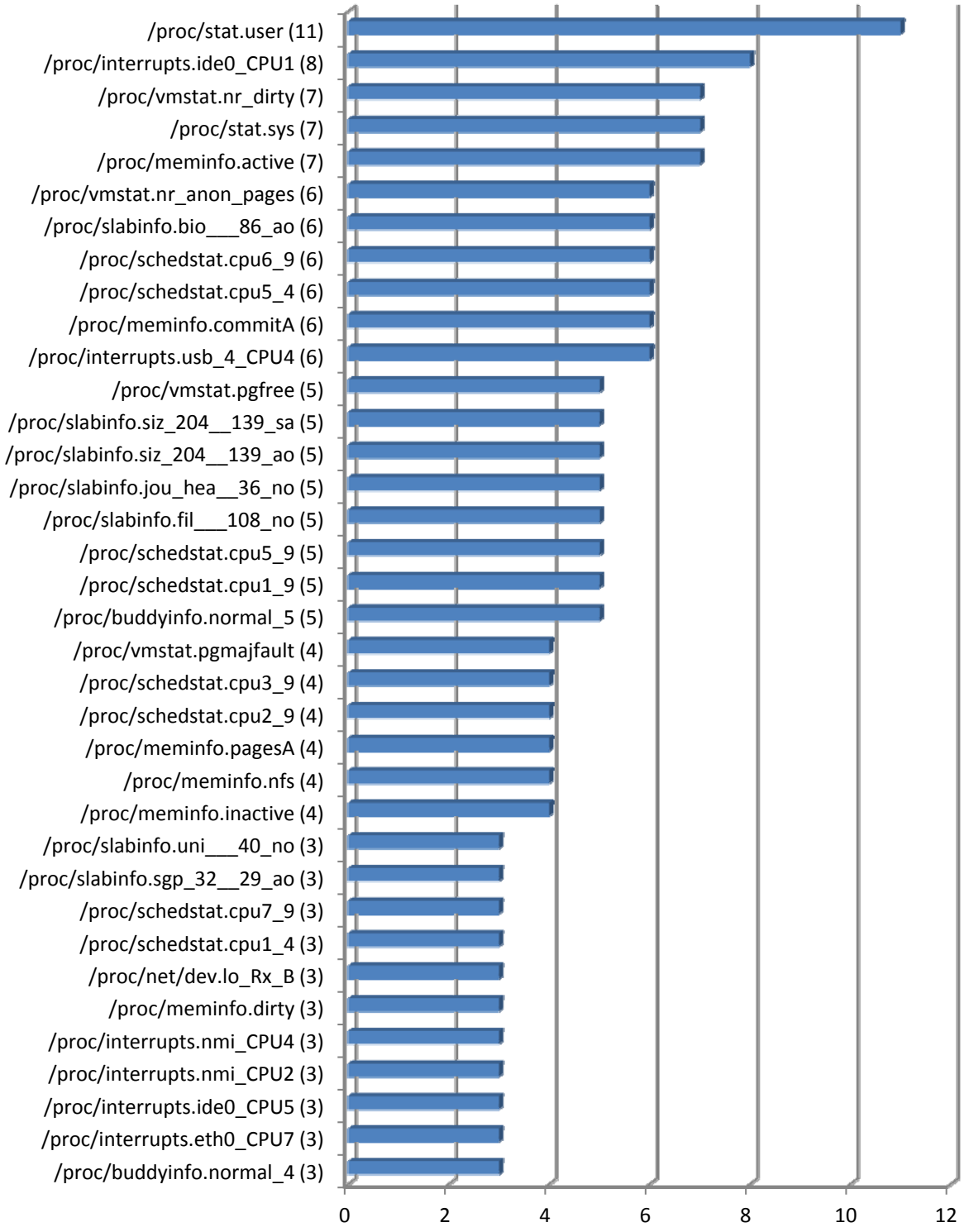


Figure 81 – *Algorithm 2* training variable frequencies (selected three or more times).

Table 69 – *Algorithm 2* training variable frequencies (selected once or twice).

Once	Twice
/proc/buddyinfo.*dma32_0*	/proc/buddyinfo.normal_0
/proc/buddyinfo.*normal_1*	/proc/diskstats.sda1_18_sec_r
/proc/buddyinfo.*normal_2*	/proc/interrupts.usb_4_CPU1
/proc/buddyinfo.*normal_7*	/proc/meminfo.pageTables
/proc/diskstats.sda_17_mer_w	/proc/schedstat.cpu4_9
/proc/diskstats.sda_17_msec_io	/proc/schedstat.cpu0_9
/proc/diskstats.sda_17_msec_w	/proc/schedstat.cpu1_6
/proc/diskstats.sda4_21_mer_r	/proc/schedstat.cpu3_4
/proc/meminfo.mapped	/proc/slabinfo.jou_hea__36_ao
/proc/schedstat.cpu0_4	/proc/slabinfo.sgp_16__30_ao
/proc/schedstat.cpu4_4	/proc/vmstat.nr_page_table_pages
/proc/schedstat.cpu5_6	/proc/vmstat.pgalloc_dma32
/proc/slabinfo.bio_1__85_ao	/proc/diskstats.sda1_18_msec_r
/proc/slabinfo.crq_poo__45_ao	/proc/interrupts.eth0_CPU6
/proc/slabinfo.siz_102__141_no	/proc/schedstat.cpu7_4
/proc/slabinfo.siz_128__150_ao	/proc/slabinfo.vm_are_s_116_no
/proc/slabinfo.siz_512__143_no	/proc/vmstat.pgalloc_normal
/proc/slabinfo.skb_fcl_c_90_ao	/proc/diskstats.sda_17_num_io
/proc/slabinfo.skb_hea_c_91_ao	/proc/interrupts.nmi_CPU0
/proc/stat.int1	/proc/interrupts.nmi_CPU1
/proc/stat.procs_r	/proc/interrupts.nmi_CPU3
/proc/vmstat.numa_hit	/proc/interrupts.nmi_CPU5
/proc/net/dev.lo_Rx_Pkt	/proc/interrupts.nmi_CPU6
/proc/stat.ctxt	/proc/interrupts.nmi_CPU7
/proc/stat.int16	/proc/net/dev.eth0_Rx_B
/proc/stat.procs	
/proc/vmstat.nr_unstable	

Table 70 – System Components and variables strongly correlated with power consumption.

System Component	Variables
processor/CPU	*user* *sys* *cpuX_9*
memory	*active*, *commitA*, *pagesA* (*nr_anon_pages*), *pageTables* (*nr_page_table_pages*), *siz_204__139_ao*, *skb_fcl_c_90_ao*, *skb_hea_c_91_ao*
network	*int1* and *eth0_CPU7*

Table 71 – System Components and variables final list.

System Component	Variables
processor/CPU	/proc/stat.*user* /proc/stat.*sys*
memory	/proc/meminfo.*commitA* /proc/meminfo.*pagesA* /proc/meminfo.*pageTables*
disk	/proc/stat.*int76*
network	/proc/stat.*int100* or /proc/stat.*int1*

Table 72 – *Model A* and *B* training results across all trials.

		Training Model A				Training Model B		
config:node		Tot. kWs	Est. kWs	% Error		Tot. kWs	Est. kWs	% Error
*n*1*p*1*c*1-t1*w*11		793.49	778.78	1.85		793.49	790.91	0.33
*n*1*p*1*c*4-t1*w*11		863.25	824.18	4.53		863.25	863.13	0.01
*n*1*p*1*c*8-*t*1*w*11		350.10	334.22	4.54		350.10	348.94	0.33
*n*1*p*1*c*8-*t*2*w*11		350.62	333.61	4.85		350.62	347.42	0.91
*n*1*p*1*c*8-*t*3*w*11		342.84	331.51	3.30		342.84	(343.68)	0.25
*n*2*p*4*c*16-*t*1*w*11		551.46	535.72	2.85		551.46	547.89	0.65
*n*2*p*4*c*16-*t*1*w*12		544.98	519.87	4.61		544.98	538.24	1.24
*n*4*p*8*c*32-*t*1*w*11		908.72	873.46	3.88		908.72	905.10	0.40
*n*4*p*8*c*32-*t*1*w*12		794.41	792.11	0.29		794.41	(796.22)	0.23
*n*4*p*8*c*32-*t*2*w*11		893.25	866.10	3.04		893.25	892.11	0.13
*n*4*p*8*c*32-*t*2*w*12		794.27	792.07	0.28		794.27	(796.00)	0.22
*n*20*p*40*c*160-*t*1*w*11		1295.75	1274.43	1.65		1295.75	1290.07	0.44
*n*20*p*40*c*160-*t*1*w*12		1222.86	1198.99	1.95		1222.86	1215.48	0.60
*n*20*p*40*c*160-*t*2*w*11		1301.19	1277.98	1.78		1301.19	1294.88	0.48
*n*20*p*40*c*160-*t*2*w*12		1222.86	1198.99	1.95		1222.86	1215.48	0.60
Average				2.76				0.45

Table 73 – Evaluation of energy estimates (*nXaX*, *Model A* and *Model B*).

	***Model A* Coefficients applied to c0-11**			***Model A* Coefficients applied to c0-12**		
config:coeffs	**Tot. kWs**	**Est. kWs**	**% Error**	**Tot. kWs**	**Est. kWs**	**% Error**
*n*2*c*16-*t*1:11	551.46	535.72	2.85	**544.98**	**535.24**	**1.79**
*n*2*c*16-*t*1:12	**551.46**	**520.30**	**5.65**	544.98	519.87	4.61
*n*2*c*16-*t*1:Avg	**551.46**	**528.01**	**4.25**	**544.98**	**527.56**	**3.20**
*n*4*c*32-*t*1:11	908.72	873.46	3.88	**794.41**	**(872.07)**	**9.78**
*n*4*c*32-*t*1:12	**908.72**	**793.35**	**12.70**	794.41	792.11	0.29
*n*4*c*32-*t*1:Avg	**908.72**	**833.40**	**8.29**	**794.41**	**(832.09)**	**4.74**
*n*4*c*32-*t*2:11	893.25	866.10	3.04	**794.27**	**(864.46)**	**8.84**
*n*4*c*32-*t*2:12	**893.25**	**793.51**	**11.17**	794.27	792.07	0.28
*n*4*c*32-*t*2:Avg	**893.25**	**829.81**	**7.10**	**794.27**	**(828.27)**	**4.28**
*n*20*c*160-*t*1:11	1295.75	1274.43	1.65	**1222.86**	**(1274.48)**	**4.22**
*n*20*c*160-*t*1:12	**1295.75**	**1199.49**	**7.43**	1222.86	1198.99	1.95
*n*20*c*160-*t*1:Avg	**1295.75**	**1236.96**	**4.54**	**1222.86**	**(1236.74)**	**1.14**
*n*20*c*160-*t*2:11	1301.19	1277.98	1.78	**1222.86**	**(1277.94)**	**4.50**
*n*20*c*160-*t*2:12	**1301.19**	**1199.53**	**7.81**	1222.86	1198.99	1.95
*n*20*c*160-*t*2:Avg	**1301.19**	**1238.76**	**4.80**	**1222.86**	**(1238.47)**	**1.28**
		$x_{c0\text{-}11}$ ***Avg***	2.64		$x_{c0\text{-}11}$ ***Avg***	**5.83**
		$x_{c0\text{-}12}$ ***Avg***	**8.95**		$x_{c0\text{-}12}$ ***Avg***	1.82
		x_{Avg} ***Avg***	**5.80**		x_{Avg} ***Avg***	**2.93**

	***Model B* Coefficients applied to c0-11**			***Model B* Coefficients applied to c0-12**		
config:coeffs	**Tot. kWs**	**Est. kWs**	**% Error**	**Tot. kWs**	**Est. kWs**	**% Error**
*n*2*c*16-*t*1:11	551.46	547.89	0.65	**544.98**	**(549.35)**	**0.80**
*n*2*c*16-*t*1:12	**551.46**	**536.42**	**2.73**	544.98	538.24	1.24
*n*2*c*16-*t*1:Avg	**551.46**	**542.16**	**1.69**	**544.98**	**543.80**	**0.22**
*n*4*c*32-*t*1:11	908.72	905.10	0.40	**794.41**	**(907.08)**	**14.18**
*n*4*c*32-*t*1:12	**908.72**	**795.65**	**12.44**	794.41	(796.22)	0.23
*n*4*c*32-*t*1:Avg	**908.72**	**850.37**	**6.42**	**794.41**	**(851.65)**	**7.21**
*n*4*c*32-*t*2:11	893.25	892.11	0.13	**794.27**	**(891.64)**	**12.26**
*n*4*c*32-*t*2:12	**893.25**	**796.06**	**10.88**	794.27	(796.00)	0.22
*n*4*c*32-*t*2:Avg	**893.25**	**844.08**	**5.50**	**794.27**	**(843.82)**	**6.24**
*n*20*c*160-*t*1:11	1295.75	1290.07	0.44	**1222.86**	**(1285.68)**	**5.14**
*n*20*c*160-*t*1:12	**1295.75**	**1218.24**	**5.98**	1222.86	1215.48	0.60
*n*20*c*160-*t*1:Avg	**1295.75**	**1254.16**	**3.21**	**1222.86**	**(1250.58)**	**2.27**
*n*20*c*160-*t*2:11	1301.19	1294.88	0.48	**1222.86**	**(1290.41)**	**5.52**
*n*20*c*160-*t*2:12	**1301.19**	**1218.46**	**6.36**	1222.86	1215.48	0.60
*n*20*c*160-*t*2:Avg	**1301.19**	**1256.67**	**3.42**	**1222.86**	**(1252.94)**	**2.46**
		$x_{c0\text{-}11}$ ***Avg***	0.42		$x_{c0\text{-}11}$ ***Avg***	**7.58**
		$x_{c0\text{-}12}$ ***Avg***	**7.68**		$x_{c0\text{-}12}$ ***Avg***	0.58
		x_{Avg} ***Avg***	**4.05**		x_{Avg} ***Avg***	**3.68**

Table 74 – Training coefficient energy estimates (*n*20*c*160*aX*-*t*2, *Models A*, *B*, and *C*).

Model A			
Node	Tot. kWs	Est. kWs	% Err.
c0-11	1301.19	1277.98	1.78
c0-12	1222.86	1198.99	1.95

Model B			
Node	Tot. kWs	Est. kWs	% Err.
c0-11	1301.19	1294.88	0.48
c0-12	1222.86	1215.48	0.60

Model C			
Node	Tot. kWs	Est. kWs	% Err.
c0-11	1301.19	1274.78	2.03
c0-12	1222.86	1197.25	2.09

Table 75 – Fitted variable energy estimates (*n*20*c*160*aX*-*t*2, *Models A*, *B*, and *C*).

		Model A				Model B				Model C		
config:coeffs		Tot. kWs	Est. kWs	% Err.		Tot. kWs	Est. kWs	% Err.		Tot. kWs	Est. kWs	% Err.
c0-11 data												
n20c160e2:11		1301.19	1277.98	1.78		1301.19	1294.88	0.48		1301.2	1274.8	2.03
n20c160e2:12		1301.19	1199.53	7.81		1301.19	1218.46	6.36		1301.2	1200.2	7.76
n20c160e2:Avg		1301.19	1238.76	4.80		1301.19	1256.67	3.42		1301.2	1237.5	4.89
c0-12 data												
n20c160e2:11		1222.86	1277.94	-4.50		1222.86	1290.41	-5.52		1222.9	1270.4	-3.89
n20c160e2:12		1222.86	1198.99	1.95		1222.86	1215.48	0.60		1222.9	1197.3	2.09
n20c160e2:Avg		1222.86	1238.47	-1.28		1222.86	1252.94	-2.46		1222.9	1233.8	-0.90

Table 76 – *Model A*, *B*, and *C* least squares coefficients.

Model A Coefficients			
Variable	$x_{c0\text{-}11}$	$x_{c0\text{-}12}$	x_{avg}
user	0.141163	0.149747	0.145455
sys	0.118702	0.120574	0.119638
int76	0.044003	0.013037	0.028520
int100	0.000685	0.000723	0.000704
commitA	0.000009	0.000005	0.000007

Model B Coefficients			
Variable	$x_{c0\text{-}11}$	$x_{c0\text{-}12}$	x_{avg}
user	0.138839	0.145449	0.142144
sys	0.116015	0.116491	0.116253
int76	0.027445	-0.005660	0.010892
int100	0.000636	0.000694	0.000665
pagesA	0.000005	0.000002	0.000004
pageTables	0.001724	0.001619	0.001672

Model C Coefficients			
Variable	$x_{c0\text{-}11}$	$x_{c0\text{-}12}$	x_{avg}
user	0.142489	0.148403	0.145446
sys	0.120030	0.120062	0.120046
int76	0.047293	0.014749	0.031021
int100	0.000661	0.000704	0.000683
pagesA	0.000009	0.000005	0.000007

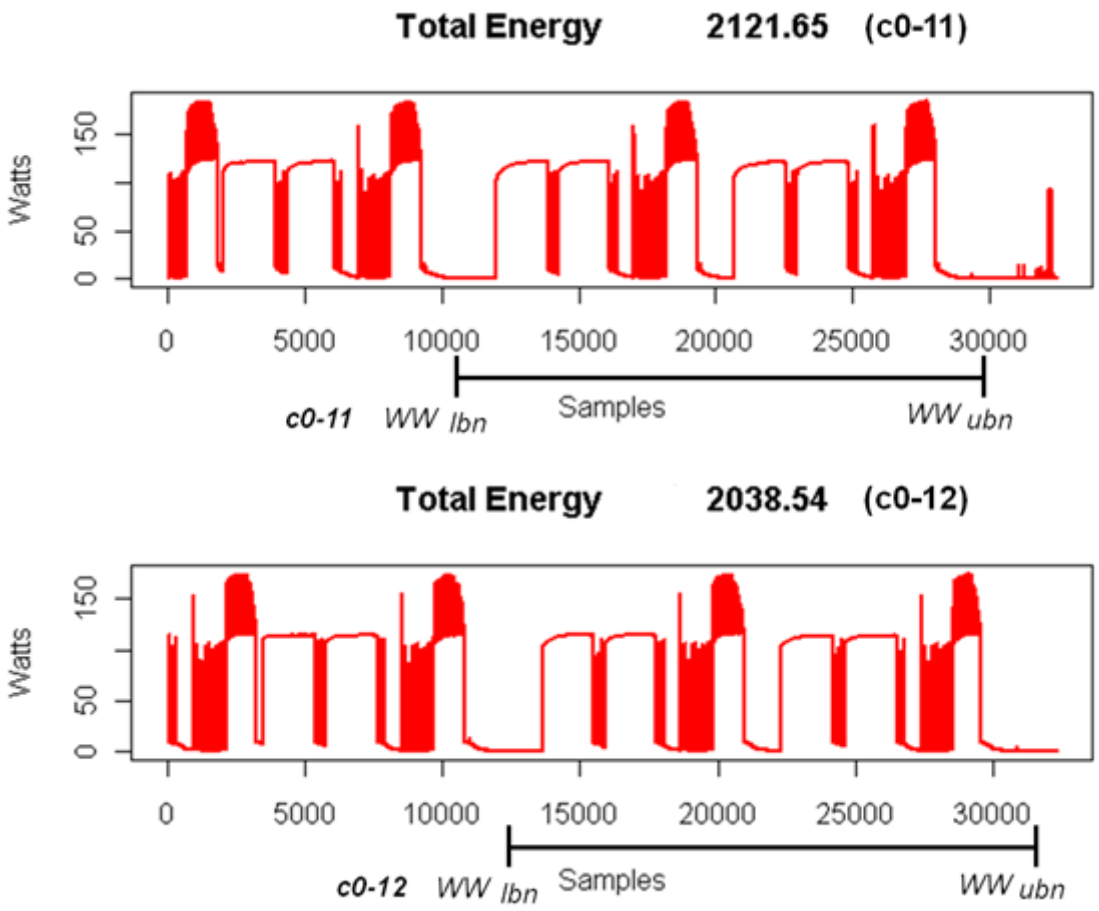


Figure 82 – Offset Watt windows for the second trial for *n*20*p*40*c*160*aX*.

Table 77 – Evaluation of modeled power consumption per compute node (*Model A*).

Model A				
	c0-11	% Err	c0-12	% Err
Tot. kWs	1301.19		1222.86	
Est. kWs	1277.98	1.78	1198.99	1.95

	$x_{c0\text{-}11}$				$x_{c0\text{-}12}$				x_{avg}			
	aligned to c0-11		aligned to c0-12		aligned to c0-11		aligned to c0-12		aligned to c0-11		aligned to c0-12	
Node	Est. kWs	% Err	Est. kWs	% Err	Est. kWs	% Err	Est. kWs	% Err	Est. kWs	% Err	Est. kWs	% Err
0	1277.18	1.845	1277.13	1.849	1198.81	7.868	1198.78	7.870	1238.00	4.856	1237.95	4.860
1	1278.93	1.711	1278.88	1.715	1199.68	7.800	1199.64	1.900	1239.31	4.756	1239.26	4.759
2	1277.11	1.851	1277.04	1.856	1198.49	7.890	1198.44	2.000	1237.80	4.872	1237.74	4.876
3	1277.57	1.815	1277.52	1.819	1198.62	7.880	1198.59	1.980	1238.10	4.849	1238.05	4.852
4	1276.01	1.935	1275.95	1.940	1197.87	7.940	1197.83	2.050	1236.94	4.938	1236.89	4.942
6	1276.50	1.897	1276.44	1.902	1198.29	7.910	1198.25	2.010	1237.40	4.902	1237.34	4.907
7	1275.17	2.000	1275.11	2.004	1197.39	7.980	1197.35	2.090	1236.28	4.989	1236.23	4.992
8	1276.36	1.908	1276.30	1.913	1197.97	7.930	1197.93	2.040	1237.16	4.921	1237.12	4.924
9	1276.28	1.914	1276.22	1.919	1198.24	7.910	1198.20	2.020	1237.26	4.913	1237.21	4.917
10	1277.00	1.859	1276.95	1.863	1198.66	7.880	1198.62	1.980	1237.83	4.869	1237.79	4.872
11	1277.98	1.784	1277.93	1.788	1199.53	7.810	1199.49	1.910	1238.76	4.798	1238.71	4.802
12	1277.98	1.784	1277.94	1.787	1199.02	7.850	1198.99	1.950	1238.50	4.818	1238.47	4.820
13	1275.64	1.964	1275.60	1.967	1197.60	7.960	1197.57	2.070	1236.62	4.962	1236.58	4.965
14	1277.07	1.854	1277.01	1.858	1198.53	7.890	1198.49	1.990	1237.80	4.872	1237.75	4.876
15	1276.11	1.927	1276.04	1.933	1198.02	7.930	1197.98	2.030	1237.07	4.928	1237.01	4.932
16	1275.74	1.956	1275.68	1.961	1197.86	7.940	1197.82	2.050	1236.80	4.949	1236.75	4.952
17	1275.58	1.968	1275.53	1.972	1197.55	7.970	1197.52	2.070	1236.56	4.967	1236.53	4.969
18	1276.24	1.917	1276.20	1.921	1198.17	7.920	1198.14	2.020	1237.21	4.917	1237.17	4.920
19	1276.30	1.913	1276.25	1.917	1198.11	7.920	1198.07	2.030	1237.20	4.918	1237.16	4.921
21	1273.40	2.136	1273.34	2.140	1195.48	8.120	1195.43	2.240	1234.44	5.130	1234.39	5.134
T/AE	**25530.15**	**1.897**	**25529.06**	**1.901**	**23963.89**	**7.915**	**23963.13**	**2.315**	**24747.04**	**4.906**	**24746.10**	**4.910**

Table 78 – Evaluation of modeled power consumption per compute node (*Model B*).

Model B				
	c0-11	% Err	c0-12	% Err
Tot. kWs	1301.19		1222.86	
Est. kWs	1294.88	0.48	1215.48	0.60

	$X_{c0\text{-}11}$				$X_{c0\text{-}12}$				X_{avg}			
	aligned to c0-11		aligned to c0-12		aligned to c0-11		aligned to c0-12		aligned to c0-11		aligned to c0-12	
Node	**Est. kWs**	**% Err**	**Est. kWs**	**% Err**	**Est. kWs**	**% Err**	**Est. kWs**	**% Err**	**Est. kWs**	**% Err**	**Est. kWs**	**% Err**
0	1292.77	0.647	1292.56	0.663	1216.48	6.510	1216.29	6.525	1254.62	3.579	1254.43	3.594
1	1290.49	0.822	1290.31	0.836	1215.32	6.599	1215.15	6.612	1252.91	3.710	1252.73	3.724
2	1291.37	0.755	1291.18	0.769	1215.45	6.589	1215.28	6.602	1253.41	3.672	1253.23	3.686
3	1290.76	0.802	1290.56	0.817	1215.29	6.602	1215.11	6.615	1253.03	3.701	1252.83	3.717
4	1290.86	0.794	1290.66	0.809	1214.70	6.647	1214.53	6.660	1252.78	3.720	1252.60	3.734
6	1291.06	0.779	1290.88	0.792	1215.02	6.622	1214.85	6.635	1253.04	3.700	1252.87	3.714
7	1291.14	0.772	1290.94	0.788	1215.10	6.616	1214.92	6.630	1253.12	3.694	1252.93	3.709
8	1297.88	0.254	1297.50	0.284	1221.42	6.131	1221.07	6.157	1259.65	3.192	1259.29	3.220
9	1300.39	0.061	1299.95	0.095	1224.16	5.920	1223.75	5.951	1262.27	2.991	1261.85	3.023
10	1291.49	0.745	1291.31	0.759	1215.41	6.592	1215.25	6.605	1253.45	3.669	1253.28	3.682
11	1294.88	0.485	1294.64	0.503	1218.46	6.358	1218.25	6.374	1256.67	3.421	1256.44	3.439
12	1290.61	0.813	1290.41	0.828	1215.65	6.574	1215.48	6.587	1253.13	3.694	1252.94	3.708
13	1289.82	0.874	1289.62	0.889	1214.62	6.653	1214.44	6.667	1252.22	3.763	1252.03	3.778
14	1289.70	0.883	1289.52	0.897	1214.63	6.652	1214.46	6.665	1252.16	3.768	1251.99	3.781
15	1289.91	0.867	1289.75	0.879	1213.79	6.717	1213.65	6.728	1251.85	3.792	1251.70	3.803
16	1290.80	0.798	1290.62	0.812	1214.92	6.630	1214.75	6.643	1252.86	3.714	1252.68	3.728
17	1289.92	0.866	1289.76	0.878	1213.82	6.715	1213.67	6.726	1251.87	3.790	1251.72	3.802
18	1297.70	0.268	1297.35	0.295	1221.30	6.140	1220.98	6.164	1259.50	3.204	1259.16	3.230
19	1301.54	-0.027	1301.11	0.006	1224.75	5.875	1224.35	5.905	1263.15	2.923	1262.73	2.956
21	1293.32	0.605	1293.01	0.629	1217.42	6.438	1217.13	6.460	1255.37	3.521	1255.07	3.544
T/AE	**25856.41**	**0.643**	**25851.64**	**0.661**	**24337.71**	**6.479**	**24333.36**	**6.496**	**25097.06**	**3.561**	**25092.50**	**3.579**

Table 79 – Evaluation of modeled power consumption per compute node (*Model C*).

Model C				
	c0-11	% Err	c0-12	% Err
Tot. kWs	1301.19		1222.86	
Est. kWs	1274.78	2.03	1197.25	2.09

	$x_{c0\text{-}11}$				$x_{c0\text{-}12}$				x_{avg}			
	aligned to c0-11		aligned to c0-12		aligned to c0-11		aligned to c0-12		aligned to c0-11		aligned to c0-12	
Node	Est. kWs	% Err	Est. kWs	% Err	Est. kWs	% Err	Est. kWs	% Err	Est. kWs	% Err	Est. kWs	% Err
0	1274.04	2.087	1274.00	2.090	1199.53	7.813	1199.50	7.815	1236.79	4.949	1236.75	4.952
1	1271.42	2.288	1271.37	2.292	1197.99	7.931	1197.96	7.934	1234.71	5.109	1234.66	5.113
2	1272.60	2.197	1272.54	2.202	1198.44	7.897	1198.40	7.900	1235.52	5.047	1235.47	5.051
3	1271.26	2.300	1271.22	2.303	1197.59	7.962	1197.55	7.965	1234.43	5.131	1234.38	5.135
4	1273.10	2.159	1273.05	2.163	1198.67	7.879	1198.63	7.882	1235.89	5.018	1235.84	5.022
6	1273.23	2.149	1273.19	2.152	1198.90	7.861	1198.87	7.864	1236.07	5.005	1236.03	5.008
7	1272.69	2.190	1272.64	2.194	1198.40	7.900	1198.37	7.902	1235.54	5.045	1235.51	5.048
8	1272.65	2.193	1272.61	2.196	1198.37	7.902	1198.33	7.905	1235.51	5.048	1235.47	5.051
9	1272.41	2.212	1272.37	2.215	1198.50	7.892	1198.47	7.894	1235.46	5.052	1235.42	5.055
10	1273.49	2.129	1273.46	2.131	1199.15	7.842	1199.12	7.844	1236.32	4.985	1236.29	4.988
11	1274.78	2.030	1274.74	2.033	1200.23	7.759	1200.20	7.761	1237.50	4.895	1237.47	4.897
12	1270.42	2.365	1270.39	2.367	1197.28	7.986	1197.25	7.988	1233.85	5.175	1233.82	5.178
13	1270.70	2.343	1270.66	2.346	1197.25	7.988	1197.22	7.990	1233.97	5.166	1233.94	5.168
14	1270.84	2.332	1270.80	2.336	1197.50	7.969	1197.47	7.971	1234.17	5.151	1234.14	5.153
15	1273.43	2.133	1273.38	2.137	1198.96	7.857	1198.93	7.859	1236.20	4.995	1236.15	4.999
16	1272.83	2.180	1272.80	2.182	1198.66	7.880	1198.63	7.882	1235.75	5.029	1235.72	5.032
17	1272.90	2.174	1272.87	2.176	1198.48	7.894	1198.45	7.896	1235.69	5.034	1235.66	5.036
18	1273.15	2.155	1273.12	2.157	1198.88	7.863	1198.85	7.865	1236.02	5.008	1235.99	5.011
19	1273.28	2.145	1273.25	2.147	1198.85	7.865	1198.82	7.867	1236.07	5.005	1236.04	5.007
21	1269.74	2.417	1269.70	2.420	1195.88	8.093	1195.84	8.096	1232.81	5.255	1232.77	5.258
T/AE	**25448.96**	**2.209**	**25448.16**	**2.212**	**23967.51**	**7.902**	**23966.86**	**7.904**	**24708.27**	**5.055**	**24707.52**	**5.058**

Table 80 – Totals of modeled power consumption per compute node (*Models A*, *B* and *C*).

c0-11 coefficients applied				
	c0-11 aligned		c0-12 aligned	
Model	Est. kWs	% Err	Est. kWs	% Err
A	25530.15	1.897	25529.06	1.901
B	25856.41	0.643	25851.64	0.662
C	25448.96	2.209	25448.16	2.212

c0-12 coefficients applied			
c0-11 aligned		c0-12 aligned	
Est. kWs	% Err	Est. kWs	% Err
23963.89	7.915	23963.13	2.315
24337.71	6.479	24333.36	6.496
23967.51	7.902	23966.86	7.904

Averaged coefficients applied			
c0-11 aligned		c0-12 aligned	
Est. kWs	% Err	Est. kWs	% Err
24747.04	4.906	24746.10	4.910
25097.06	3.561	25092.50	3.579
24708.27	5.055	24707.52	5.058

Table 81 – *Models A*, *B* and *C* training results (*n*20*p*40*c*160*aX*-*t*2).

Metered Node: c0-11					Metered Node: c0-12			
Model A	**x**	**Est. kWs**	**Est. kWs%**		**Model A**	**x**	**Est. kWs**	**Est. kWs%**
user	0.141163	193.34	15.13		*user*	0.149747	202.97	16.93
sys	0.118702	776.49	60.76		*sys*	0.120574	790.47	65.93
int76	0.044003	1.04	0.08		*int76*	0.013037	0.27	0.02
int100	0.000685	88.75	6.94		*int100*	0.000723	93.11	7.77
commitA	0.000009	218.37	17.09		*commitA*	0.000005	112.16	9.35
Node Est.		1277.98			**Node Est.**		1198.99	
Cluster Est.		25559.63			**Cluster Est.**		23979.81	

Metered Node: c0-11					Metered Node: c0-12			
Model B	**x**	**Est. kWs**	**Est. kWs%**		**Model B**	**x**	**Est. kWs**	**Est. kWs%**
user	0.138839	190.16	14.69		*user*	0.145449	197.15	16.22
sys	0.116015	758.91	58.61		*sys*	0.116491	763.71	62.83
int76	0.027445	0.65	0.05		*int76*	-0.005660	-0.12	-0.01
int100	0.000636	82.38	6.36		*int100*	0.000694	89.30	7.35
pagesA	0.000005	124.13	9.59		*pagesA*	0.000002	36.64	3.01
pageTables	0.001724	138.66	10.71		*pageTables*	0.001619	128.81	10.60
Node Est.		1294.88			**Node Est.**		1215.48	
Cluster Est.		22094.39			**Cluster Est.**		20366.61	

Metered Node: c0-11					Metered Node: c0-12			
Model C	**x**	**Est. kWs**	**Est. kWs%**		**Model C**	**x**	**Est. kWs**	**Est. kWs%**
user	0.142489	195.16	15.31		*user*	0.148403	201.15	16.80
sys	0.120030	785.17	61.59		*sys*	0.120062	787.12	65.74
int76	0.047293	1.11	0.09		*int76*	0.014749	0.31	0.03
int100	0.000661	85.58	6.71		*int100*	0.000704	90.64	7.57
pagesA	0.000009	207.76	16.30		*pagesA*	0.000005	118.04	9.86
Node Est.		1274.78			**Node Est.**		1197.25	
Cluster Est.		25495.61			**Cluster Est.**		23945.02	

Table 82 – Modeled power consumption per variable averaged across all compute nodes.

Coefficients Applied: c0-11					Coefficients Applied: c0-12					Coefficients Applied: Average			
Model A	**x**	**Est. kWs**	**Est. kWs%**		**Model A**	**x**	**Est. kWs**	**Est. kWs%**		**Model A**	**x**	**Est. kWs**	**Est. kWs%**
user	0.141163	190.89	14.95		user	0.149747	202.50	16.90		user	0.145455	196.69	15.90
sys	0.118702	778.35	60.97		sys	0.120574	790.62	65.98		sys	0.119638	784.49	63.40
int76	0.044003	1.08	0.08		int76	0.013037	0.32	0.03		int76	0.028520	0.70	0.06
int100	0.000685	88.45	6.93		int100	0.000723	93.39	7.79		int100	0.000704	90.92	7.35
commitA	0.000009	217.74	17.06		commitA	0.000005	111.37	9.29		commitA	0.000007	164.56	13.30
Node Est.		1276.51					1198.20					1237.35	
Clstr. Est.		25530.13					23963.90					24747.02	

Coefficients Applied: c0-11					Coefficients Applied: c0-12					Coefficients Applied: Average			
Model B	**x**	**Est. kWs**	**Est. kWs%**		**Model B**	**x**	**Est. kWs**	**Est. kWs%**		**Model B**	**x**	**Est. kWs**	**Est. kWs%**
user	0.138839	187.74	14.57		user	0.145449	196.68	16.22		user	0.142144	192.21	15.37
sys	0.116015	760.73	59.03		sys	0.116491	763.85	63.00		sys	0.116253	762.29	60.95
int76	0.027445	0.67	0.05		int76	-0.005660	-0.14	-0.01		int76	0.010892	0.27	0.02
int100	0.000636	78.04	6.06		int100	0.000694	85.09	7.02		int100	0.000665	81.57	6.52
pagesA	0.000005	123.23	9.56		pagesA	0.000002	36.98	3.05		pagesA	0.000004	80.11	6.41
pageTables	0.001724	138.34	10.73		pageTables	0.001619	129.94	10.72		pageTables	0.001672	134.14	10.73
Node Est.		1101.02					1015.73					1058.37	
Clstr. Est.		22020.41					20314.55					21167.48	

Coefficients Applied: c0-11					Coefficients Applied: c0-12					Coefficients Applied: Average			
Model C	**x**	**Est. kWs**	**Est. kWs%**		**Model C**	**x**	**Est. kWs**	**Est. kWs%**		**Model C**	**x**	**Est. kWs**	**Est. kWs%**
user	0.142489	192.68	15.14		user	0.148403	200.68	16.75		user	0.145446	196.68	15.92
sys	0.120030	787.06	61.85		sys	0.120062	787.27	65.69		sys	0.120046	787.16	63.72
int76	0.047293	1.16	0.09		int76	0.014749	0.36	0.03		int76	0.031021	0.76	0.06
int100	0.000661	85.29	6.70		int100	0.000704	90.91	7.59		int100	0.000683	88.10	7.13
pagesA	0.000009	206.26	16.21		pagesA	0.000005	119.16	9.94		pagesA	0.000007	162.71	13.17
Node Est.		1272.45					1198.38					1235.41	
Clstr. Est.		25448.99					23967.53					24708.26	

Table 83 – *virgo2* compute node utilization samples fitted to c0-11 Watt meter samples.

1301.19		Model A			Model B			Model C	
Node		**Est. kWs**	**% Err.**		**Est. kWs**	**% Err.**		**Est. kWs**	**% Err.**
0		1278.01	1.78		1294.44	0.52		1274.91	2.02
1		1279.40	1.67		1295.12	0.47		1275.50	1.97
2		1280.19	1.61		1295.21	0.46		1276.49	1.90
3		1279.42	1.67		1295.12	0.47		1275.46	1.98
4		1279.78	1.65		1294.76	0.49		1276.05	1.93
6		1279.90	1.64		1294.71	0.50		1275.98	1.94
7		1279.95	1.63		1294.80	0.49		1276.13	1.93
8		1279.83	1.64		1296.24	0.38		1275.74	1.96
9		1279.44	1.67		1296.91	0.33		1275.55	1.97
10		1279.22	1.69		1294.01	0.55		1275.28	1.99
11		1277.98	1.78		1294.88	0.48		1274.78	2.03
12		1279.03	1.70		1295.31	0.45		1275.16	2.00
13		1279.27	1.68		1294.81	0.49		1275.43	1.98
14		1279.34	1.68		1294.63	0.50		1275.34	1.99
15		1279.81	1.64		1294.23	0.53		1276.00	1.94
16		1279.89	1.64		1294.58	0.51		1275.98	1.94
17		1279.92	1.63		1294.68	0.50		1275.99	1.94
18		1279.46	1.67		1295.94	0.40		1275.54	1.97
19		1279.14	1.69		1296.93	0.33		1275.37	1.98
21		1279.30	1.68		1295.39	0.45		1275.30	1.99
T/AE		25588.28	1.68		25902.70	0.45		25511.98	1.99

Table 84 – *virgo2* compute node utilization samples fitted to c0-12 Watt meter samples.

1222.86		Model A			Model B			Model C	
Node		**Est. kWs**	**% Err.**		**Est. kWs**	**% Err.**		**Est. kWs**	**% Err.**
0		1200.01	1.87		1217.24	0.46		1197.98	2.03
1		1199.36	1.92		1215.34	0.61		1197.56	2.07
2		1200.85	1.80		1217.22	0.46		1198.64	1.98
3		1200.32	1.84		1215.47	0.60		1198.02	2.03
4		1200.37	1.84		1216.18	0.55		1198.09	2.03
6		1200.61	1.82		1216.76	0.50		1198.19	2.02
7		1200.56	1.82		1216.77	0.50		1198.25	2.01
8		1200.76	1.81		1218.34	0.37		1198.38	2.00
9		1200.87	1.80		1218.76	0.34		1198.51	1.99
10		1200.22	1.85		1216.86	0.49		1197.97	2.04
11		1199.92	1.88		1217.16	0.47		1197.81	2.05
12		1198.99	1.95		1215.48	0.60		1197.25	2.09
13		1200.27	1.85		1216.66	0.51		1198.00	2.03
14		1200.36	1.84		1214.98	0.64		1198.10	2.02
15		1200.81	1.80		1216.65	0.51		1198.50	1.99
16		1200.49	1.83		1216.51	0.52		1198.12	2.02
17		1200.55	1.82		1217.05	0.48		1198.15	2.02
18		1200.46	1.83		1218.61	0.35		1198.15	2.02
19		1199.52	1.91		1219.14	0.30		1197.02	2.11
21		1200.36	1.84		1218.02	0.40		1198.04	2.03
T/AE		24005.66	1.84		24339.20	0.40		23960.73	2.03

Table 85 – *Model A, B, C* variables fitted to c0-11.

Model A	c0-11	Average	Std	Est. kWs Avg.	Est. kWs Avg. %
user	0.141163	0.122753	0.007448	166.02	12.97
sys	0.118702	0.122390	0.002860	802.54	62.71
int76	0.044003	0.043955	0.021721	1.75	0.14
int100	0.000685	0.001878	0.006373	55.57	4.34
commitA	0.000009	0.000011	0.000001	253.86	19.84
Model B	**c0-11**	**Average**	**Std**	**Est. kWs Avg.**	**Est. kWs Avg. %**
user	0.121737	0.120136	0.007400	162.48	12.54
sys	0.116974	0.119663	0.002643	784.66	60.58
int76	-0.010919	0.026292	0.020004	0.89	0.07
int100	0.000503	0.001164	0.003437	48.63	3.75
pagesA	0.000008	0.000007	0.000001	164.28	12.68
pageTables	0.001599	0.001675	0.000077	134.26	10.37
Model C	**c0-11**	**Average**	**Std**	**Est. kWs Avg.**	**Est. kWs Avg. %**
user	0.142489	0.123548	0.007439	167.10	13.10
sys	0.120030	0.123527	0.002588	809.99	63.48
int76	0.047293	0.048252	0.022903	1.95	0.15
int100	0.000661	0.001995	0.007056	51.25	4.02
pagesA	0.000009	0.000011	0.000001	245.72	19.26

Table 86 – *Model A, B, C* variables fitted to c0-12.

Model A	c0-12	Average	Std	Est. kWs Avg.	Est. kWs Avg. %
user	0.134198	0.132099	0.006802	178.62	14.88
sys	0.119103	0.122400	0.003136	802.58	66.85
int76	0.019372	0.035550	0.024342	1.59	0.13
int100	0.000686	-0.000382	0.004179	67.75	5.64
commitA	0.000006	0.000006	0.000001	150.06	12.50
Model B	**c0-12**	**Average**	**Std**	**Est. kWs Avg.**	**Est. kWs Avg. %**
user	0.131635	0.129095	0.006363	174.56	14.34
sys	0.116257	0.119201	0.003101	781.61	64.22
int76	0.001016	0.016473	0.019301	0.67	0.05
int100	0.000637	-0.001181	0.007573	62.83	5.16
pagesA	0.000003	0.000003	0.000001	66.27	5.45
pageTables	0.001610	0.001637	0.000072	131.09	10.77
Model C	**c0-12**	**Average**	**Std**	**Est. kWs Avg.**	**Est. kWs Avg. %**
user	0.135215	0.132436	0.006222	179.08	14.94
sys	0.123101	0.122981	0.003100	806.39	67.29
int76	0.005772	0.037821	0.025030	1.69	0.14
int100	0.000540	-0.000362	0.004003	65.40	5.46
pagesA	0.000006	0.000006	0.000001	145.84	12.17

Table 87 – Coefficients and estimated energy for *Model D*.

Model D	x	Est. kWs
user	0.018532	27.57
sys	0.007757	52.43
int76	0.253472	6.16
int100	-0.000368	-48.21
pagesA	0.000077	1894.41
pageTables	0.047709	4057.36
commitA	-0.000194	-4951.25

Vita

Mario Caire received his bachelor's degree in Psychology from The University of Texas at Austin in December 1992 and his Master of Science degree in Electrical and Computer Engineering from The University of Texas at El Paso in the summer of 2003. He immediately began doctoral studies the following semester at UTEP also in Electrical and Computer Engineering and began research in the area of power modeling of high performance computer clusters in 2008. Mario's work experience includes teaching, database administration, system administration, network administration, web application development, web programming, and social science research with a focus on regional economic development. He has been employed at UTEP since December 2000 where in January 2007 he began working for the Institute for Policy and Economic Development as a programmer analyst and is currently a system analyst and researcher for IPED. Mario's research interests include high performance computing, highly parallel and data intensive computing, distributed programming, agent based modeling, industry cluster analysis, and regional economic development.

Permanent address: 601 Maple
El Paso, Texas 79903

This thesis/dissertation was typed by Mario E. Caire.