DETAILED LOW-COST ENERGY AND POWER MONITORING OF COMPUTING SYSTEMS

By

Chad Michael Paradis

B.S. University of Maine, 2013

A THESIS

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Master of Science

(in Computer Engineering)

The Graduate School

The University of Maine

August 2015

Advisory Committee:

Vincent Weaver, Assistant Professor, Advisor

Bruce Segee, Professor

Yifeng Zhu, Professor

THESIS ACCEPTANCE STATEMENT

On behalf of the Graduate Committee for Chad Michael Paradis, I affirm that this manuscript is the final and accepted thesis. Signatures of all committee members are on file with the Graduate School at the University of Maine, 42 Stodder Hall, Orono, Maine.

Vincent Weaver, Assistant Professor (Date)

LIBRARY RIGHTS STATEMENT

In presenting this thesis in partial fulfillment of the requirements for an advanced degree at The University of Maine, I agree that the Library shall make it freely available for inspection. I further agree that permission for "fair use" copying of this thesis for scholarly purposes may be granted by the Librarian. It is understood that any copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Chad Michael Paradis (Date)

DETAILED LOW-COST ENERGY AND POWER MONITORING OF COMPUTING SYSTEMS

By Chad Michael Paradis

Thesis Advisor: Vincent Weaver

An Abstract of the Thesis Presented
in Partial Fulfillment of the Requirements for the
Degree of Master of Science
(in Computer Engineering)
August 2015

Power and energy are increasingly important metrics in modern computing systems. Large supercomputers utilize millions of cores and can consume as much power as a small town; monitoring and reducing power consumption is an important task. At the other extreme, power usage of embedded and mobile devices is also critically important. Battery life is a key concern in such devices; having detailed power measurement allows optimizing these devices for power as well.

Current systems are not set up to allow easy power measurement. There has been much work in this area, but obtaining power readings is often expensive, intrusive, and not well validated. In this work we propose a low-cost, easy-to-use, power measurement methodology that can be used in both high-end servers and low-end embedded systems. We then validate the results gathered against existing power measurement systems. We extend the existing Linux perf utility so that it can provide real-world fine-grained power measurements, allowing users easy access to these values, enabling new advanced power optimization opportunities.

DETAILED LOW-COST ENERGY AND POWER MONITORING OF COMPUTING SYSTEMS

By Chad Michael Paradis

Thesis Advisor: Vincent Weaver

A Lay Abstract of the Thesis Presented
in Partial Fulfillment of the Requirements for the
Degree of Master of Science
(in Computer Engineering)
August 2015

Keywords: power, energy, super computing, cluster computing

Power and energy are increasingly important metrics in modern computing systems. Large supercomputers utilize millions of cores and can consume as much power as a small town; monitoring and reducing power consumption is an important task. At the other extreme, power usage of embedded and mobile devices is also critically important. Battery life is a key concern in such devices; having detailed power measurement allows optimizing these devices for power as well.

Current systems are not set up to allow easy power measurement. There has been much work in this area, but obtaining power readings is often expensive, intrusive, and not well validated. In this work we propose a low-cost, easy-to-use, power measurement methodology that can be used in both high-end servers and low-end embedded systems. We then validate the results gathered against existing power measurement systems. We extend the existing Linux perf utility so that it can provide real-world fine-grained power measurements, allowing users easy access to these values, enabling new advanced power optimization opportunities.

DEDICATION

To my grandparents.

ACKNOWLEDGEMENTS

I would like to thank Vince, my advisor, for being there to help me with any and all problems during my time with him and for being so patient. I would also like to thank the rest of the faculty and staff in the ECE department including Tony for giving me some much needed breaks by talking with me during some late nights, and Cindy and Janice for being ever helpful and cheerful in the office. I would like to give a special thanks to Yifeng, Bruce, Andy, and Rick for helping me build the foundation for everything I learned in my time at the University of Maine. I would also like to thank all my classmates for their help over the years and my family and friends for supporting me.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION AND MOTIVATION

Energy and power consumption are increasingly important design constraints in modern computing systems, yet gathering information on these metrics remains difficult. This work describes the instrumentation and analysis of energy and power on modern systems with the intention of gaining insight into how these machines consume energy.

There are many existing frameworks for gathering power and energy usage of a system. Typically they have low resolution (1Hz or less), are expensive (requiring costly calibrated voltage meters), do not provide data in real time (data is gathered for later analysis), are not fully validated, and are not easily obtained.

This research looks into providing high-resolution, low-cost, validated, well-documented power meters that can be easily procured and installed into systems allowing detailed, fine-grained analysis of power and energy.

1.1 Motivation

Motivations for this kind of power and energy measurement are varied. In general it is always good to save power and energy, if only because this saves the consumer money when it is time to pay the power bill. Reducing power consumption also helps in other ways; often, reduced power correlates to reduced heat generation, which can help with cooling costs and fan noise, and other side effects. A group with the most to gain is those who have large numbers of computers, including large data centers and supercomputers. Some of the fastest supercomputers in the world [5] have millions of processors and use many Megawatt s of power (an amount equal to the consumption of a small town). By just saving 1 watt per core you can save 1 megawatt of electricity which is a major accomplishment.

Another area where power savings matter are embedded or mobile devices (one common example is a cell phone). Saving power in this case will directly correspond to longer battery life, or the possibility for using smaller, more lightweight batteries so there is a large interest in conserving power and energy in this field too.

Power measurement can also provide benefits more than just reducing overall power bills. One problem is not the total amount of power, but using a constant amount. Cameron [14] gives an example where a large cluster of Xeon/Tesla machines can have power swings of as much as 62% within 50ms in some workloads. If many machines have swings like this at once it can put a large strain on the power distribution network. Being able to monitor and maybe avoid such large swings can also help when implementing large computer systems.

1.2 Background

Power and energy are commonly used terms that have specific meanings. They are often used interchangeably, but the terms mean different things. Power describes instantaneous electrical usage whereas energy describes the total amount of electrical work done over a period of time.

1.2.1 Power

Power is most generally defined as the rate of doing work. In an electrical system the power is defined as the rate of transmission of electricity in a circuit and is measured in watts. Power usage is important to consider when designing an electrical system because it dictates the nature of the power supply. Exceeding power supply limits may result in system failure.

Instantaneous power is defined in Equation 1.1 as the relationship between voltage, current, and resistance, in accordance with Ohm's Law.

$$P = V \cdot I = I^2 \cdot R = \frac{V^2}{R} \tag{1.1}$$

To measure power experimentally, two of the three values must be known. For a complex system such as a CPU the resistance cannot be directly measured. This means that the input voltage and the current must be used to calculate the power.

1.2.2 Energy

Power is not sufficient to describe the electrical usage of a system. Power simply provides an instantaneous snapshot of the system and since computational tasks run for long periods, time must also be considered. Where power describes the instantaneous electrical usage of a system energy describes the electrical usage over time and can most easily be seen in an electrical bill at the end of the month.

Average energy over a time period can be found by multiplying the average power over that time by the length of the time period as shown in Equation 1.2.

$$E = W \cdot t \tag{1.2}$$

More accurately, energy can be described as the integral of power over time as shown in Equation 1.3. This relationship between power and energy is essential when designing instrumentation. It is as important to get accurate timestamps as it is to get accurate power measurement data.

$$E = \int_0^t P dt \tag{1.3}$$

1.2.3 Energy Delay

Simply optimizing for energy does not necessarily provide the best performance of a system. Using very low energy is useless if the system takes a long time to provide results. Various attempts have been made to provide a metric that takes into account the conflicting demands of fast response time and low energy consumption. The most popular variants are some variation on the Energy Delay Product [37, 54].

The aim of energy delay is to provide a single metric for comparing a computational task's energy usage and execution time. This is accomplished by adding another time factor into the typical energy equation. Depending on the importance of run time versus energy the time factor can be raised to a power to highlight its contribution to the metric. Equation 1.4 shows this relationship.

$$ED = E \cdot delay^n \tag{1.4}$$

1.3 Power and Energy Measurement

Power and energy are important metrics but generally are overlooked in favor of execution time when optimizing for performance. This is likely due to an almost universal lack of easily accessible metrics provided to the user. Program run time can be measured directly but measuring the power and energy usage for a given computing task is not as straightforward.

A myriad of related factors must be considered:

- Should static overheads such as cooling be included?
- Should peripherals such as disk usage and network I/O be included?
- Should operating system overhead be included?

When measuring power and energy at the system level at a low temporal resolution it can be difficult to isolate all of the various factors that impact measurement. Finer resolution power data can be acquired by measuring subsystems individually (when possible) and by sampling at much higher data rates.

CHAPTER 2
RELATED WORK

Power and energy have been design concerns for a long time, and there is much related work. My research contribution is in constructing a tool that can provide low-cost fine-grained measurements to the user in real-time. Results are also compared to and validated against other existing off-the-shelf measurement tools and integrated power measurement metrics.

The related work has two primary areas: projects that measure power (with validation) and existing tools that provide power readings.

2.1 Power Measurement

There are various existing ways to determine the energy and power used by computing systems. One way is by direct measurement of the current flowing into the various components. Often it is not possible to instrument each part of a system (or any part at all) so work has been done to estimate the power based on related performance metrics. Even when power is estimated, studies typically also do some manner of actual measurement as a way of validating their models.

There are various components that combine to create the total power used in a system. This includes the CPU, the RAM, any disks, GPUs, network adapters, and any other I/O. In addition power distribution, power-supply inefficiency, and cooling also consume power. Previous work has looked both at the power consumption of individual components as well as full-system power.

In my work I design an infrastructure that measures the actual power used by systems (no estimation is involved) at a fine granularity using inexpensive instrumentation. The CPU, memory, and total power are measured (although not all of the systems allow measuring all of the values). This information is then provided to the user to allow program power optimization.

2.1.1 CPU Power Measurement

CPU power is difficult to measure directly as the lines connecting the power supply to the CPU are often not directly broken out and thus hard to access. For this reason it is often more common to estimate CPU power usage rather than directly measure it. A model for CPU power is often constructed from hardware performance metrics.

According to some Intel documentation [40], the 4-conductor auxiliary (P4) 12V ATX connector is dedicated to the CPU Voltage. In this case the CPU power can be estimated by measuring this connector, and this is a common way of measuring CPU power. A more destructive way of measuring power involves tapping into the voltage regulator circuitry or even cutting traces on the motherboard to insert sense resistors.

Joseph and Martonosi [44] estimate power of a Pentium Pro processor; Contreras and Martonosi [17] estimate power on ARM, and Wu et al. [77] do this for Pentium 4. Goel et al. [30] and Singh, Bhadauria and McKee [70] look at using machine learning to create accurate CPU power models using performance counter results.

Some modern CPUs have the capability of generating power estimates on the fly, based on performance counter results, thermal measurements, and other factors. This includes Intel's Running Average Power Limit (RAPL) [19, 66, 42] and AMD's Application Power Management (APM) [7], and similar functionality on IBM's Power7 [26]. These readings are only estimates, but some work has been done to validate them showing reasonable accuracy [21, 34, 66].

Molka et al. [58] use a ZES LMG450 power meter sampling at 10Hz to measure the power consumption of individual x86-64 instructions, although they are limited by extracting the CPU power from full-system measurements (they do not monitor the CPU power directly).

Porier et al. [64] describe the "Foxton Technology" on an Itanium processor that has an embedded processor with four A/D converters that measure temperature and power (using an on-die resistor) every $8\mu s$. The measurements are used for frequency scaling and are not exposed to the user.

Mesa-Martinez et al. [56] measure the temperature of a CPU in detail using an oil bath and an infrared camera. They then estimate the power consumption based on thermal measurements. They validate this against power readings taken with a multi-meter (compensating for a 10% loss in the CPU voltage regulator circuitry). Hamann et al. [35] do similar work with estimating power from temperature.

2.1.2 DRAM Power

The power consumption of main memory is of interest, although usually it is overlooked since it tends to be much smaller than the power used by the CPU. Power is not often measured; more common is to simulate it and use values from datasheets to provide a basis for the models. When actual power is measured it is usually done by modifying a DIMM extender board to have a sense resistor.

Gottscho et al. [31, 32] measure DRAM power using a 2Ω sense resistor and a digital multimeter sampling at 10 samples/second. Rahmati et. al [65] measure DRAM power using a 200Ω sense resistor and an A/D board.

DIMM power can also be estimated, often by utilizing CPU cache miss information. Intel RAPL can provide DRAM power estimates on some machines (primarily servers). Contreras et al. [17] estimate DRAM energy on ARM processors.

Schmidt and Wehn [67] measure the power consumption of DRAM on an embedded board and compare it against existing simulators and models.

2.1.3 GPU Power

On modern computing systems the graphics processor (or GPU) can use a large amount of power. This is especially true of machines used for gaming, as well as supercomputers where GPGPU work is done (large calculations done on the graphics device).

Measuring the power involves intercepting the PCIe power lanes and measuring the current. Some GPUs, such as high-end NVIDIA cards, support the NVML library [61] which can provide power measurements. Also Intel RAPL can also provide GPU power estimates for GPUs integrated into the CPU package.

In this work I do not measure the GPU power.

2.1.4 Hard Drive Power

Storage devices also consume power. In typical desktop systems the power usage is overshadowed by the CPU power, but in situations with large number of disks (such as a RAID arrays) the power consumption can be significant. Measuring the power often involves intercepting the power connectors as they come into the disk; hard drives are a lot easier to instrument for power than most of the other hardware components. It is also possible to estimate power usage. Often operating system metrics are used to determine when I/O happens, and models are constructed of the various moving parts in the drive based on actual measurements.

Hylick et al. [39] measure in detail power consumption of 10 different hard drives by placing .02Ω resistors on the power lines and logging 12-bit values at 1100Hz. Lee et al. [52] measure power consumption using Hall Effect sensors at 16-bit resolution. Yan et al. [78] measure power of hard drives using the DEEP/LEEP framework [69, 68] which provides samples at 10kHz.

In this work I do not measure hard drive power values.

2.1.5 Full-system Measurements

The easiest way to obtain full-system measurements is to measure the A/C power at the wall outlet. Off-the-shelf tools such as the watts up? PRO (WUP) [23] meter can yield a decent estimate for total system power usage but fail to expose subsystem measurements. The WattsUpPro also has poor temporal resolution, only recording one measurement per second.

Some server nodes provide similar low-frequency power measurements via the IPMI subsystem, which can often be accessed either remotely or using helper utilities as described in Section 4.4.3.

I break the full-system measurement methods out into those of server systems and those of embedded systems.

2.1.5.1 Servers

The Powerpack [28] project describes a fully instrumented x86 cluster. PowerMon2 [10] involves small boards that can measure 8-channels of the ATX power supply at high resolution. Hackenberg et al. [34] describe many cluster measurement techniques, including RAPL.

LEAP [69] is most similar to my work. They instrument an Intel Atom low-end machine and gather power measurements of the CPU (by tapping into the power converter), DRAM (with a DDR2 extender and sense resistor), and hard drive (.01Ω sense resistor), and provide results over the serial port that can be time correlated with execution time. It uses a dedicated A/D converter for providing the measurements at 10kHz.

Cui et al. [18] instrument a full system. They measure disk (with a 0.02 ohm resistor and a 50x amplifier) at 20us resolution. They also measure the CPU via the 12V lines, DIMMs with an 0.02 ohm resistor, and the network and video cards with a PCI extender. They use a digital multimeter to gather results.

Power on servers can also be estimated. Economou et al. [22] look at performance in a server; they use micro-benchmarks to determine rough power consumption and then generate estimates for the various components in a system (CPU, RAM, hard-drive, etc.) Lee and Brooks [51] use machine learning to predict the power usage of a full system.

Bircher and John [12] generate models to estimate full system behavior using performance counters. They look at estimating Disk, DRAM, and I/O power usage based on various CPU counters and validate using various benchmark suites (SPEC2k, dbt-2, SPECjbb). They use sense resistors for instrumentation with the data recorded for later analysis, synchronized with a pulse over serial port.

Various papers look at the various ATX wires in an attempt to find out which powers what component [28, 24, 15]. Chen et al. [15] use a multimeter to measure CPU power (using the dedicated ATX line) and the "brown wires" on the ATX connector to measure memory and a WattsUpPro for full system power measurement. Feng et. al [24] measure the power consumption on a cluster. They instrument the machines at the ATX level with 0.1 Ohm resistors and logging multimeters. They find the CPU powered by 5V pins, memory and others through 3.3V pins, and the 12V pins primarily for fans. Castaño et. al [59] measure the 12V ATX power supply lines and use those to create a model for predicting full system behavior, including CPU and disk activity. Khoshbakht and Dimopoulos [46] investigate system power, measuring current on the ATX CPU power connector with an acquisition device sampling at 15k samples/second (which is then smoothed) and then comparing against the MARS86 simulator. Diouri et al. [20] validate a variety of power meters that measure A/C power (OmegaWatt, WattsUpPro?) and ones that measure ATX power lines (PowerMon2, NI and DCM). They use the pmlib framework. They find the external meters give similar results, but the internal ones vary with

PowerMon2 giving different results than the others. Piga et al. [63] use hall-effect sensors to investigate in detail the power consumption of all of the ATX lines.

Mahesri and Vardhan [53] break down the power consumption of a laptop using an oscilloscope and a clamping current probe.

The IBM BlueGene series of supercomputers is based off of an embedded design and often ranks highly in the Top Green 500 supercomputers list [4]. The BluegGene P and Q systems provide an interface for measuring system power [79, 74, 36].

Laros et al. [49] utilize the i2c connection to the voltage regulator board on Cray computers to gather detailed power information.

Economou et al. [22] propose Mantis which is a detailed full-system power estimation environment that is based on results gathered from an instrumented server blade. They measure the various power planes, and split the 12V plane to add a sense resistor to enable splitting of memory and CPU values.

Hsu and Poole [38] describe power measurement details for supercomputing clusters starting with the utility connection down to the individual processors.

In this work not only do I measure the ATX power on desktop systems, but I also measure the power on the custom HP connector. I have not found any reference of others attempting to measure power on this common type of server power supply.

2.1.5.2 Embedded Systems

Power is not only important for servers, but also in embedded systems. In my work I restrict my measurements to desktop and server machines, but the same measurement tools I develop could be used on embedded systems.

Typically on embedded systems it is hard to access the internal power lines (unless the board designer provided test points). Therefore you are usually limited to full-system measurements by tapping into the input power line. Some boards, such as the Beagleboard, provide a sense resistor connected to a built-in A/D converter allowing power measurements. However most embedded boards have a System-on-Chip (SoC) design where as much functionality as possible is included in a single chip, so breaking out detailed powered measurements is difficult.

Stanley-Marbell and Cabezas [73] compare Beagleboard, PowerPC, and x86 low-power systems for thermal and power. Aroca et al. [9] compare Pandaboard, Beagleboard, and various x86 boards and measure FLOPS/W. Jarus et al. [43] compare the power and energy efficiency of Cortex-A8 systems to x86 systems. Laurenzano et al. [50] compare Cortex A9, Cortex A15 and Intel Sandybridge and measure power and performance on a wide variety of HPC benchmarks.

Cloutier et al. [16] investigate power and performance of a wide variety of ARM boards as well as on a cluster built of Raspberry Pi nodes. These nodes are instrumented to measure power, by having a sense resistor across the input power which is amplified by an opamp and measured by an SPI A/D converter.

2.1.6 Profiling Tools

In addition to measuring power, I provide a tool that allows gathering real time power information along with other performance info.

There are many tools that provide power information. Many of them are just reading the Intel RAPL values, often in conjunction with the PAPI [13, 75] library.

In this work I modify the perf utility to provide actual power readings alongside the RAPL values that perf can already access.

2.1.6.1 RAPL based tools

A number of tools are capable of reading and generating energy results by accessing the Intel RAPL counters. Khan et al. describe IgProf [45] which uses PAPI and RAPL.

Moghaddam et al. [57] summarize some different tools used for gathering RAPL and IPMI data in a production environment.

Pmlib [6] is a power measurement utility used by various groups when gathering power results from A/D converters.

2.1.6.2 Power and Energy Application Programming Interfaces

The Intel Energy Checker SDK [41] provides a software interface allowing access to various power meters, including the WattsUpPro.

Pmlib [6] is a power measurement utility used by various groups when gathering power results from A/D converters.

PowerScope [25] is a tool that allows optimizing code for energy use on embedded platforms. It provides an interface that allows placing calipers around code and then gathering power information via an external digital volt meter.

Knapp et al. [47] use an existing tool called PerfTrack when investigating the impact of cooling technologies on power consumption of supercomputers.

Laros et al. [48] propose the PowerAPI which is designed to let applications gather power information from all levels of the hardware stack and environment.

CHAPTER 3
INSTRUMENTATION AND SOFTWARE DESIGN

This chapter covers the design and construction of the custom instrumentation, the serial embedded logging device, and the modification of the perf utility to allow for real-time transmission of data.

3.1 Instrumentation

The instrumentation can be divided into two main categories: high power instrumentation and low current instrumentation. Both types of instrumentation use different methods in an attempt to accurately measure the current through the system while minimizing the impact on the system. The complete system schematic is given in Figure 3.1. Both methods are used in conjunction to gain information about different components in a system.

3.1.1 High Power Measurements

High voltage lines are the primary transmission lines in a modern system. These are likely 12V lines that transmit power from the power supply to the various subsystems which regulate the 12V signal down to a more useful level. A higher voltage transmission line means lower losses over a length of wire and allows for smaller gauge wires. These high power lines can transmit up to 700W and in excess of 50A. The high power measurement circuit takes these characteristics into consideration and can be seen in Figure 3.2.

When measuring high-power (and thus high-current) lines it is often best to use Hall effect sensors. Hall effect sensors measure current indirectly by sensing the magnetic field generated by an electric signal. To the circuit being measured,

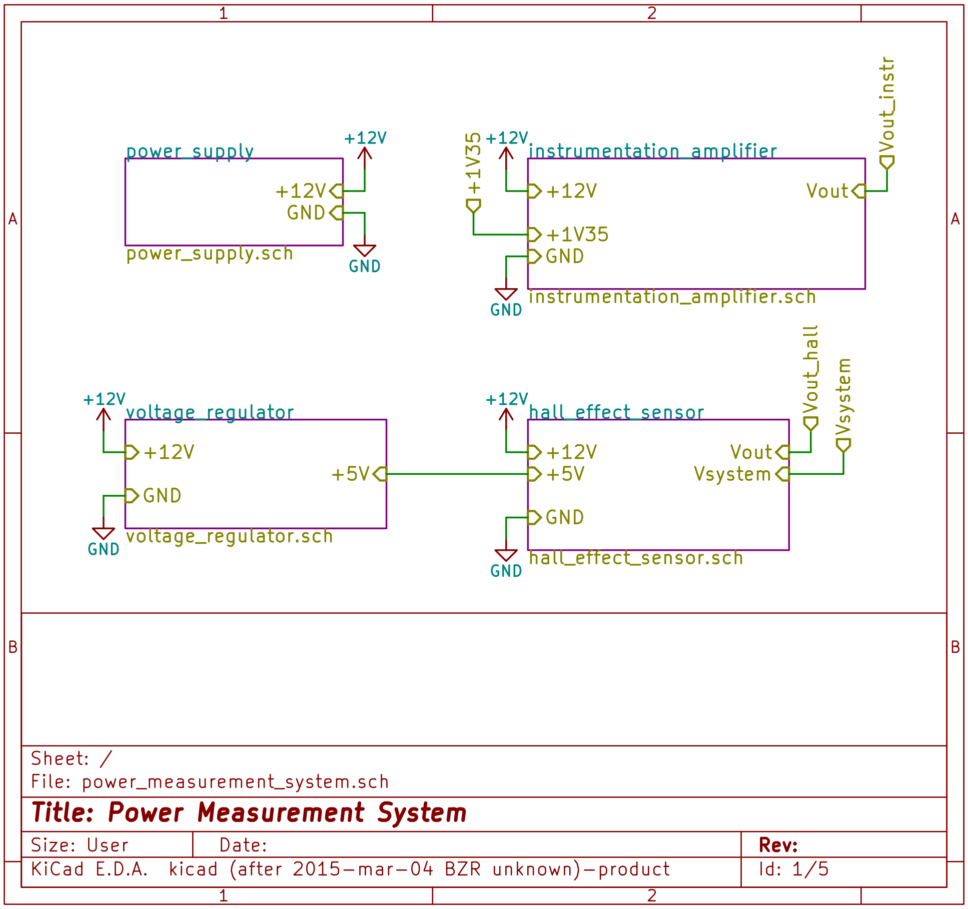


Figure 3.1: Power Measurement System

the sensor looks like a very small resistor on the order of 1mΩ in series with the system. The sensing circuitry outputs a voltage proportional to the magnitude of the magnetic field. This allows very high currents to be measured with minimal impact on the system. The circuit employs an Allegro MicroSystems Hall Effect Sensor (ACS715) for current measurements which requires a 5V supply and can measure DC currents up to 30A. As the test systems can have currents up to 60A, two ACS715 are used.

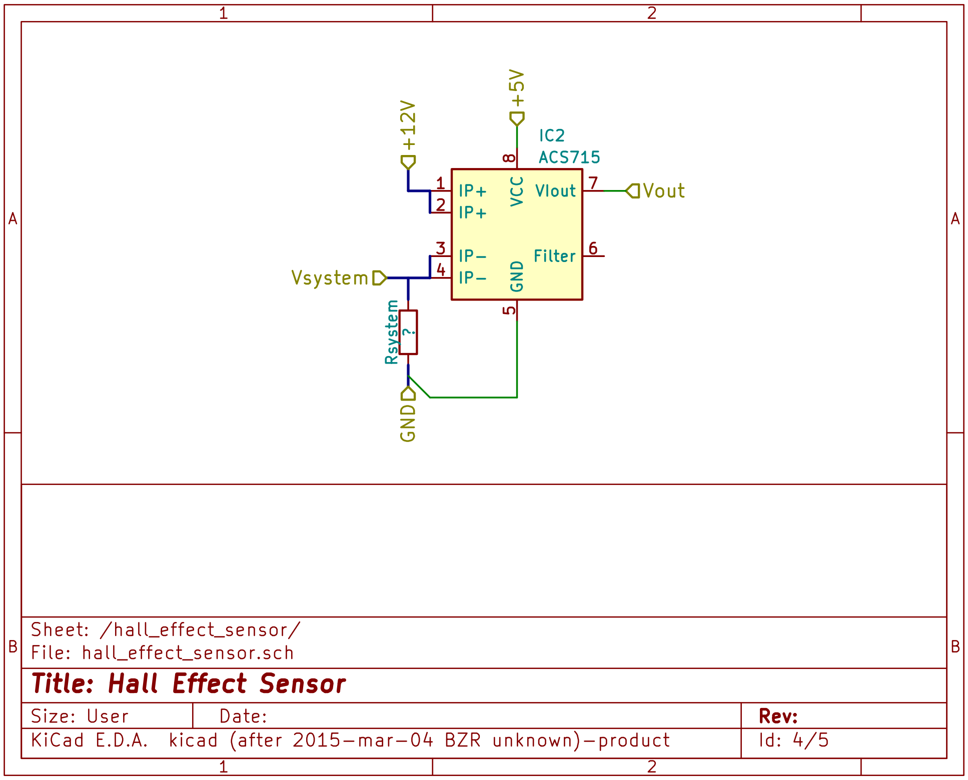

1
2
A
B
+5V
+12V
IC2
ACS715
8
1
2
3
4
5
6
7
IP+
IP+
IP−
IP−
VCC
GND
Vlout
Filter
Vout
Vsystem
Rsystem
?
GND
Sheet: /hall_effect_sensor/
File: hall_effect_sensor.sch
Title: Hall Effect Sensor
Size: User
Date:
Rev:
KiCad E.D.A. kicad (after 2015−mar−04 BZR unknown)−product
Id: 4/5

Figure 3.2: Hall Effect Sensor Circuit

The high power measurement system uses quick disconnect electrical connectors in order to connect the Hall effect sensors to various power supplies. The quick disconnects offer a simple, flexible way to connect the measurement circuitry up to any instrumented power supply.

3.1.2 Low Power Measurements

There are various subsystems in a computer that use relatively little power (this includes the memory system, the disk drive, network card, and several others). For example, common DRAM power consumption is around 5W with a 1.35V supply. This means the current through the RAM is approximately $3.7A = 5.0W/1.35V$.

With such a low current the DRAM line is not suitable to be measured by the Hall effect sensors mentioned in Section 3.1.1. The resolution of the Hall effect sensors is too low to accurately gauge low currents. Instead, a small value power resistor and an instrumentation amplifier is used to measure the current. The power resistor is placed in series with the system and a voltage drop proportional to the current can be seen across it. An instrumentation amplifier is used to amplify the small differential signal to levels that the embedded system can accurately read. An instrumentation amplifier is similar to an op-amp differential amplifier, but with additional stages added to buffer the signal and avoid impedance mismatch. The circuit can be seen in Figure 3.3.

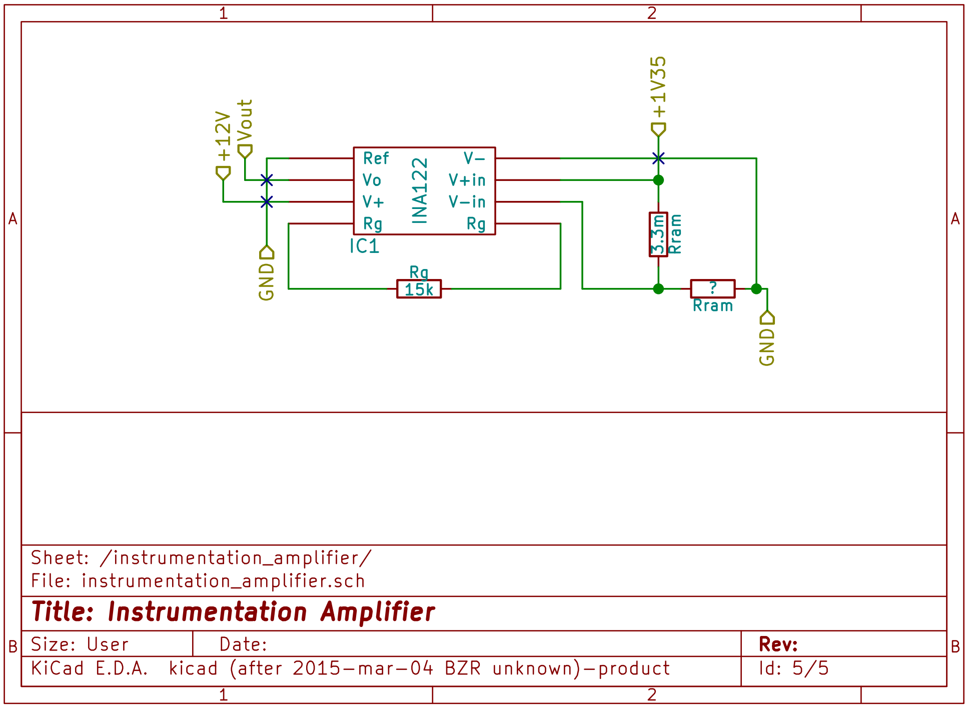

1
2
A
B
+12V
Vout
GND
Ref
Vo
V+
Rg
INA122
V−
V+in
V−in
Rg
IC1
Rg
15k
+1V35
3.3m
Rram
?
Rram
GND
Sheet: /instrumentation_amplifier/
File: instrumentation_amplifier.sch
Title: Instrumentation Amplifier
Size: User
Date:
Rev:
KiCad E.D.A. kicad (after 2015−mar−04 BZR unknown)−product
Id: 5/5

Figure 3.3: Instrumentation Amplifier Schematic

This low power measurement circuit uses a small value power resistor and an Burr-Brown Instrumentation Amplifier (INA122) to measure the voltage drop across the power resistor. The INA122 runs on a wide range of supply voltages and has an easily configurable gain via a single resistor. The INA122 accepts two differential inputs and can output an amplifier result. The gain is set to be 18.3 with a 15kΩ resistor. The gain equation is given in Equation 3.1 The ease-of-use flexibility were the determining factors for choosing the INA122.

$$G = 5 + \frac{200000}{R_G} \tag{3.1}$$

3.2 Power Supplies

The source of all power in a computing system is the power supply which converts wall-outlet high-voltage alternating current electricity to the low-voltage direct current electricity used internally. In this work we instrument two common types of power supplies: Advanced Technology eXtended (ATX) power supplies and Hewlett-Packard (HP) Common Slot (CS). ATX supplies are most commonly used in desktops and CS power supplies are used in some server systems.

3.2.1 ATX Power Supply

One of the most commonly used power supplies in computing is the ATX power supply. The ATX specification was designed by Intel in 1995 to simplify power supply design and as such ATX supplies are prevalent on most modern day desktops. Figure 3.4 shows the common 24-pin connector and the accompanying 4-pin connector that is often used to provide large amounts of power to the CPU or GPU. Only the 4-pin ATX connector is instrumented for these tests, as previous work [40, 28, 15] has shown that the 4-pin connector powers the CPU and the other power lines on the 24-pin connector are not used for much.

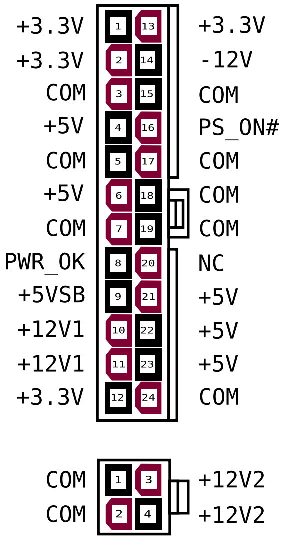


Figure 3.4: 24-Pin and 4-Pin ATX Connectors

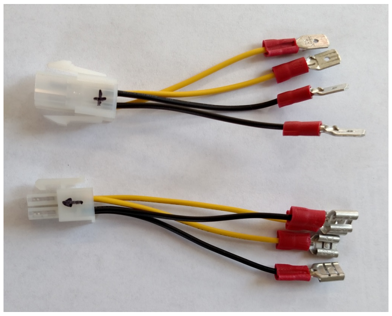


Figure 3.5: 4-Pin Advanced Technology eXtended Connector Instrumentation

3.2.2 Hewlett-Packard Common Slot Power Supply

The instrumentation uses the quick disconnects mentioned in Section 3.1.1. The ATX P4 connector was cut and quick disconnects were crimped to all eight connectors which allows the 12V lines to connected to the Hall effect circuitry. One of the ground signals is also passed into the circuitry so the ATX power supply and measurement system share a common reference. These modifications are shown in Figure 3.5.

The CS power supply provides several benefits over the ATX supply, including redundancy and ease of replacement in the case of a failure. The drawback of the CS design is that solid metal contacts are used instead of wires and as such are much harder to access and instrument to measure power and energy. Figure 3.6 shows these solid contacts.

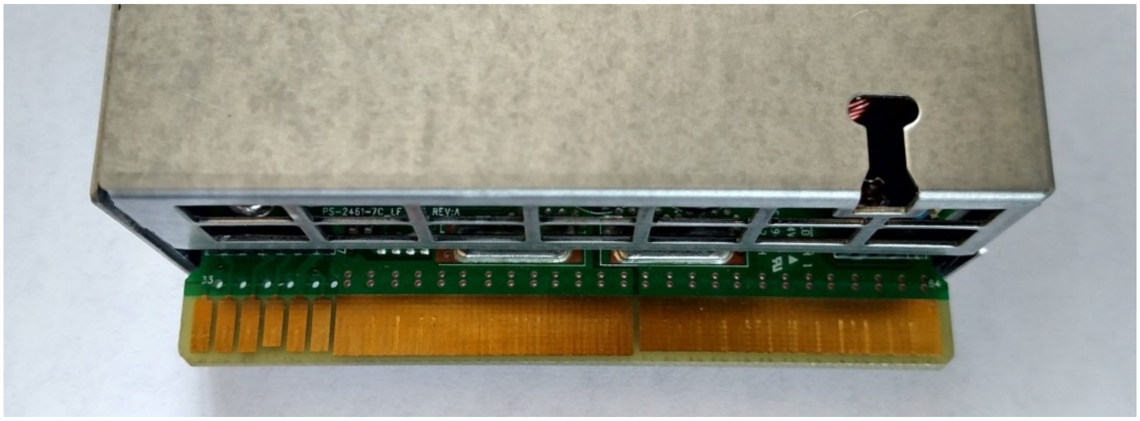


Figure 3.6: HP Common Slot Power Supply Contacts

The power supplies used are capable of providing up to 700W of power at 60A. Instrumentation was designed with these specifications in mind. The compact form factor of the CS power supply is space efficient and ideal for a dense server rack. A CS power supply is shown in Figure 3.7

3.2.2.1 Common Slot Extender

In order to tap into these inaccessible power lines, special extender boards were designed and built. The extender boards allow a CS power supply to be plugged into the female end and the male end of the extender to be plugged into the server rack as normal. The extender board breaks out all power and signal lines in the CS power supply. The extender board PCB layout is shown in Figure 3.8.

The extender board is designed to handle high currents on the 12V and ground lines. Each high current line has four 16AWG wires that can handle up to 80A. The

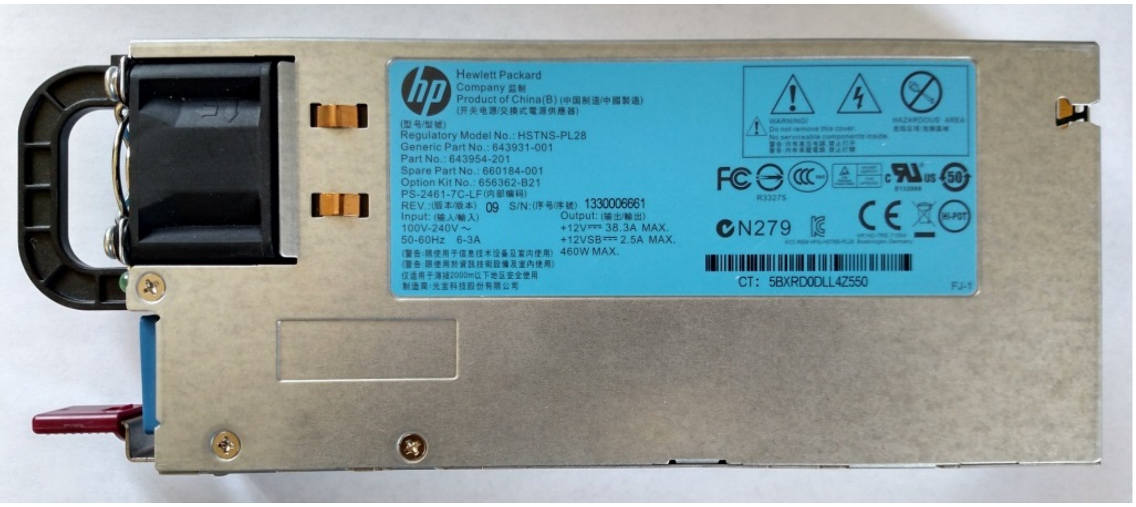

Hewlett Packard
Company 监制
Product of China(B) (中国制造/中國製造)
(开关电源/交換式電源供應器)
(型号/型號)
Regulatory Model No.: HSTNS-PL28
Generic Part No.: 643931-001
Part No.: 643954-201
Spare Part No.: 660184-001
Option Kit No.: 656362-B21
PS-2461-7C-LF(内部编码)
REV.:(版本/版本) 09 S/N:(序号/序號) 1330006661
Input: (输入/輸入)
100V-240V ~
50-60Hz 6-3A
Output: (输出/輸出)
+12V 38.3A MAX.
+12VSB 2.5A MAX.
460W MAX.
(警告:限使用于信息技术设备及室内使用)
(警告:限使用於資訊技術設備及室內使用)
仅适用于海拔2000m以下地区安全使用
制造商:光宝科技股份有限公司
WARNING!
Do not remove this cover.
No serviceable components inside.
HAZARDOUS AREA
R33275
N279
CT: 5BXRD0DLL4Z550
FJ-1

Figure 3.7: HP Common Slot Power Supply

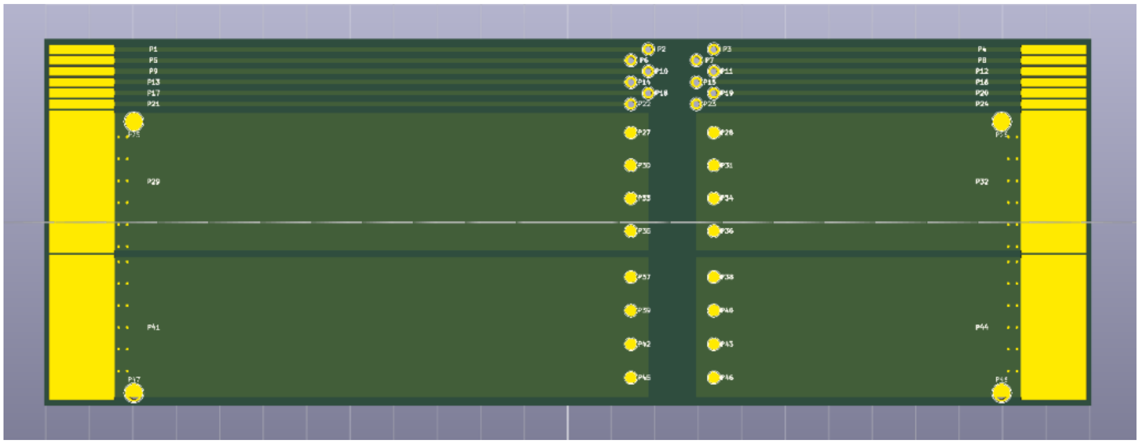


Figure 3.8: HP Common Slot Extender

traces themselves are 32mm wide and can handle up to 25A each. With two traces per channel and two channels (one on each side of the power supply contacts) this allows for up to 100A to be transmitted. A complete table of current considerations is given in Table 3.1.

The CS power supply contacts were also duplicated to allow the extender to be plugged directly into a female CS power supply slot. This extender tongue and the signals on each contact can be seen in Figure 3.9. Note that extender tongue is

Table 3.1: Current Considerations

Features	Dimension	Current Per Feature	Number
Traces	32mm	25A	4
Vias	.5mm	3.5A	20
Through-Hole	1.5mm	7.5A	8
Wire	16 AWG	18A	8

identical in layout on both sides with varying pin functions. The important signals to recognize are the large 12V and GND signals. The 12V lines transport a majority of the power to the system motherboard while the GND signal is needed as a reference. The boards were custom designed using KiCad [1] and fabricated by OSHPark.

Something else to consider is the length of the extender board. The board is just over ten inches long to allow room for power measurement wires to come out of the power supply slot. Such a length means that the power supply rests completely outside the power supply slot and does not block access to the slot.

3.3 DRAM Instrumentation

The DRAM instrumentation uses the low power measurement circuits discussed in Section 3.1.2 as well as a JET-5464 DDR3 DIMM extender. The JET-5464 is a simple pass through extender with the addition of current sensing resistors across the supply lines. The power resistor measures 3.33mΩ which is small enough to not influence the supply voltage significantly. A 10A current through the resistor results in a .0333V drop across the resistor which is likely small enough to be ignored by most systems. Leads were connected to measure the voltage drop across the sense resistor as well as to the voltage reference of the DIMM. A picture of the extender and leads is shown in Figure 3.10.

Originally, a JET-5452 DIMM DDR3 extender was modified to allow power measurement but it turns out to be very difficult to cut the traces and connect to

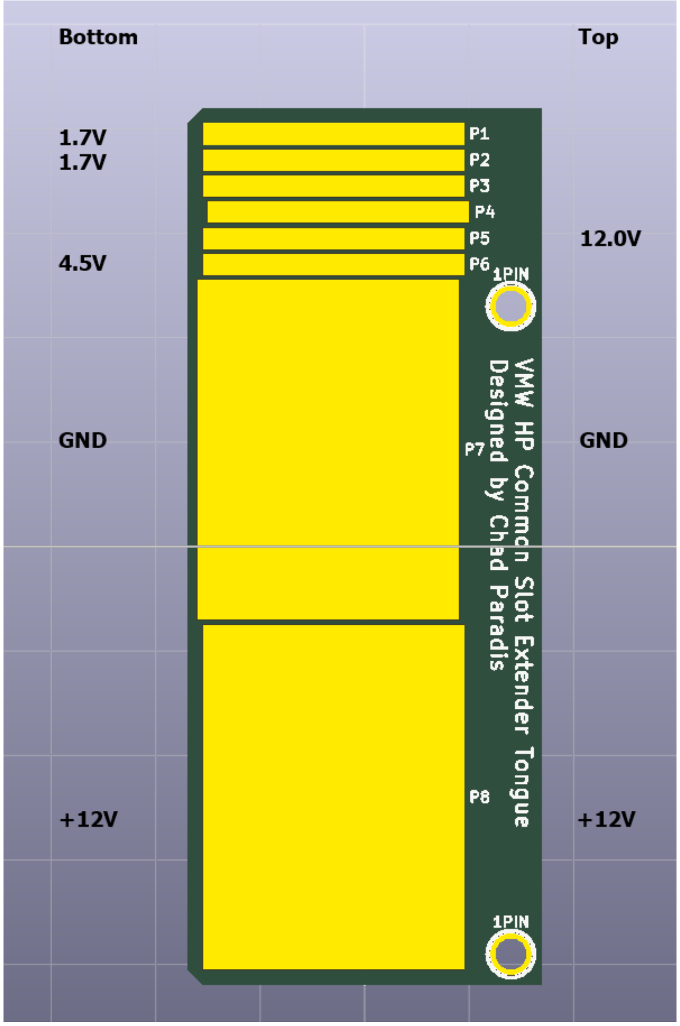


Figure 3.9: HP Common Slot Tongue

all 22 power lines to the low power measurement system. In the end it was much easier to buy the already instrumented DIMM extender.

3.4 Integrated and Off-the-Shelf Solutions

For comparison, total system power is gathered with a WUP as well as Intelligent Plaform Management Interface (IPMI).

The WUP functions as a standalone solution by sitting between the wall outlet and the power supply. Once plugged in, the WUP gathers data automatically which

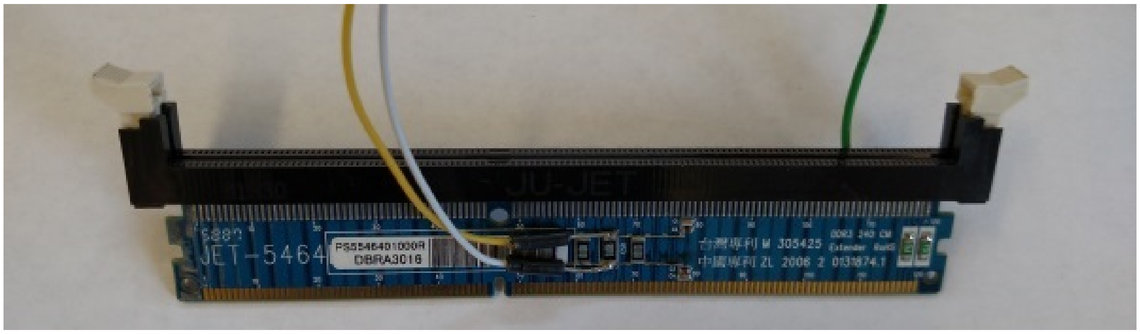


Figure 3.10: JET-5464 DDR3 DIMM Extender

can be downloaded with an easy-to-use Windows interface called the Watts Up USB Data Logger. The WUP can be configured to log data about a variety of metrics including current, voltage, power, and energy. The data can be saved to a CSV format with the Windows USB interface.

IPMI is an integrated system that appears on a system-by-system basis. Two of the test systems include IPMI functionality. IPMI data can be queried from a remote system or locally using the ipmitool under Linux. If querying remotely, a username and password are required. Querying locally requires no authentication. Requesting IPMI power readings is as simple as executing `ipmitool -c sensor get ’Power Meter’` in a shell. Some sample output that might be seen while benchmarking is seen in Figure 3.11. Note that an extra timestamp is prepended to the sample of data. This is added by a bash script described in Section 4.4.3 and is used to synchronize the IPMI data with the various other sources.

```
1433796529,679545361
Locating sensor record...
Power Meter,156,Watts,ok,7.9,Device Enabled
```

Figure 3.11: IPMI Output

Table 3.2: Table of Test Systems perf Features

Architecture	RAPL CPU	RAPL RAM	APM	IPMI
Haswell	YES	YES	NO	NO
Deneb	NO	NO	NO	NO
Sandy Bridge	YES	NO	NO	YES
Piledriver	NO	NO	YES	YES

3.4.1 Estimated CPU Power

Information about estimated CPU power is gathered from Running Average Power Limit (RAPL) and Application Power Management (APM) interfaces. Both methods offer ways to gain insight into CPU power usage but their methods differ.

A complete table of the test systems and their features is shown in Table 3.2

3.5 Serial Power Logging Device

To facilitate real-time power and energy feedback from a system, an embedded system with USB-to-serial capabilities was used. Initially, an STM32F4 development board was used but when it was found that the STM32F4 did not have enough analog-to-digital converter (ADC) channels for future tests it was substituted for a Teensy 3.1 [3]. The main program logic is shown in Figure 3.12.

The main loop is a simple client-server model with the Teensy as the server and the benchmarked system as the client. The client sends commands to the server and The server responds with the requested information or by executing the requested actions.

3.5.1 Analog-to-Digital Converter

The Teensy ADC features two separate ADC modules. These are able to sample simultaneously for maximum performance. Overall, 20 channels are available for use while only 10 are easily accessible. The ADC has a 16-bit resolution at over 300kHz and is accessed through a third party library.

```
while (1) {
    request = Serial.read();

    switch (request) {
        case MY_SERIAL_INT:
            interval = get_interval();
            break;
        case MY_SERIAL_CHX:
            channels = get_channels();
            break;
        case MY_SERIAL_BEG:
            begin_sample(interval, channels, samples);
            break;
        case MY_SERIAL_REQ:
            transmit_sample(channels, samples);
            break;
        case MY_SERIAL_TBO:
            nbo = test_network_byte_order();
        case MY_SERIAL_END:
            break;
        case MY_SERIAL_NUL:
        default:
            break;
    }
}
```

Figure 3.12: Teensy 3.1 Main Program Logic

3.5.2 USB-to-Serial Communication

In order to transmit data in real-time a USB-to-Serial library was used. The library emulates a serial port on the client side and allows the server to easily read and write data. The serial device can be accessed from Linux with the termios library as a normal serial port. Communication is at 115200 baud with an 8n1 data format.

3.5.3 Embedded System Configuration

The embedded system is configurable in two ways; The sampling interval and channel list can be configured with commands from the client. MY_SERIAL_INT is used to specify the interval. The list of channels to gather data about can be specified by sending an integer containing a bitmap to the embedded device. The bits in the integer correspond to the channels to sample with bit 0 mapping to channel 0 and so on. For example, Figure 3.13 shows an example of how to configure the embedded system with channels defined as int channels = 0b00001101; (channels 1, 3, and 4).

```
int length = 0, bytes_written = 0;
char channel_string[64];

length = sprintf(channel_string, "%c%i", MY_SERIAL_CHX, my_channels);
do {
    length -= bytes_written;
    bytes_written = write(my_tty_fd, channel_string + bytes_written, length);
} while (bytes_written < length);
```

Figure 3.13: Sending a Channel Bitmap

3.5.4 Miscellaneous Concerns

Although the power lines of most modern supplies can be up to 12V, it is almost never a concern while measuring them with the custom instrumentation that they will exceed the 3.3V max of the Teensy's ADC. The Hall effect sensor, however, will output nearly 4V when measuring a 30A supply line. This was an oversight that could lead to problems and could possibly damage the Teensy's ADC pins and as such it may be necessary to step down the output of the Hall effect sensors in order to protect the Teensy and ensure accurate readings. The 12V signal is also stepped down to get an accurate voltage reference for the power calculation.

3.6 The Linux Perf Tool

The main performance monitoring tool used under Linux is the perf tool [29]. Perf is included in the Linux kernel's main source code tree under the tools directory. The primary use of perf is to read hardware performance counters and display them to the user; a common use case is to examine branch and cache miss behavior of a program. Many software counters are also available that give information about the operating system.

Perf allows for process specific metrics and can be used to analyze a specific program or to hook into a running process. The results may reveal that a program should be rewritten or optimized to improve cache or branch behavior.

The perf tool is modified so that in addition to the standard performance counters, it can also provide real-time power measurements from the serial power measurement infrastructure.

3.6.1 Compiling Perf

Detailed instructions for compiling the Linux kernel are readily available so an abbreviated list of steps necessary to compile and run a custom version of perf are given. To compile a custom version of the perf tool on Debian use instructions similar to those in Listing 3.14. These instructions assume the instrumentation repository is in the same directory as the root of the Linux kernel source.

```
wget https://www.kernel.org/pub/linux/kernel/v3.x/linux
   -3.18.13.tar.xz
tar xf linux-3.18.13.tar.xz
cd linux-3.18.13/tools/perf
mv builtin-stat.c builtin-stat.c.bak
ln -s ../../../instrumentation/power_measurement_system/
   simple_serial_program
ln -s simple_serial_program/src/builtin-stat.c
ln -s simple_serial_program/src/ssplib.c
cd util
ln -s ../simple_serial_program/util/my_defines.h
ln -s ../simple_serial_program/util/ssplib.h
cd ..
make perf
```

Figure 3.14: Compiling Perf

```
    perf stat ./stream
```

Figure 3.15: Basic perf example

```
perf stat -I100 -a -e /power/energy-cores/ mpiexec -np 24 xhpl
```

Figure 3.16: Typical perf Usage

3.6.2 Using Perf

Using perf is straightforward. A simple use case is shown in Figure 3.15.

The perf tool can take a large set of options which enable gathering information about other performance counters, output the results to a file, print results periodically, and many other useful things.

A typical usage of perf in this research is given in Figure 3.16. For the serial code to be executed the interval option must be specified. RAPL counters are specified with `-a -e /power/energy-cores/` and `mpiexec -np 24 ./xhpl` is the benchmark to be run. The `-a` flag is used to tell the perf tool to look at system wide performance counters which is needed for RAPL counters to function The resulting performance counters and serial data will be printed directly to standard out.

3.6.3 Perf Patch

For this research the perf tool was modified and code was inserted in strategic locations in order to gather and associate real-time power readings with perf's performance counters. The patch can be found in Appendix A.

In order to avoid modifying kernel makefiles and streamline the compilation process all functions were written in their own source file and included in `builtin-stat.c`. This means that the library is compiled when `builtin-stat.c` is so no dependencies are added to the perf makefile. Serial initialization is done in the `__run_perf_stat()` function. The serial device file is opened, the baud rate and buffering options, and the embedded system itself is configured. The embedded system is told to start polling when perf enters the `handle_initial_delay()` function. Samples are requested and printed to standard out in the `print_counter_aggr()` function.

CHAPTER 4
BENCHMARKS AND EXPERIMENTAL METHODOLOGY

Four machines were instrumented with custom hardware, benchmarked, and measured for power and energy. The custom instrumentation is used to measure power usage in tandem with integrated and off-the-shelf solutions including WUP, APM, RAPL, and IPMI

4.1 Test Systems

A variety of test systems were used for experimentation with a range of architectures and features, including a mix of high-end server systems and consumer desktops. Server systems often have additional administrative features via embedded interfaces such as Integrated Lights Out (iLO) and IPMI. These interfaces allow for remote management of the system and in some instances allow for monitoring power and energy usage. Two of the systems used have this capability.

Note that all tests were run with Dynamic Voltage Frequency Scaling (DVFS) and Turbo modes enabled. This introduces more unpredictability into the tests as far as the CPU is concerned but it also gives more realistic results as most systems in practice will have these options enabled.

Modern architectures also include a variety of performance counters that allow measurement of various metrics such as: instructions per second, instructions per cycle, branch miss rate, and a variety of others. In recent years manufacturers have also added performance counters that expose information about energy usage in various parts of the system. Intel RAPL uses performance counters and a finely tuned mathematical model to estimate the energy usage of their CPUs and CPU subsystems. RAPL allows measurements of CPU components such as the complete

Table 4.1: Table of Test Systems

Architecture	Model	Clock	Threads	RAM
Haswell	Intel i5-4570S	2.90GHz	4	4GB
Deneb	AMD Phenom II X4 955	3.20GHz	4	2GB
Sandy Bridge	Intel E5-2640	2.50GHz	24	8GB
Piledriver	AMD Opteron 6376	2.30GHz	32	8GB

CPU package, CPU core, and DRAM. AMD APM system is comparable but uses hardware measurements to estimate power usage.. An overview of the test machines is given in Table 4.1

4.1.1 Desktop Systems

4.1.1.1 Haswell

The Haswell system has an Intel i5-4570S Haswell processor with a 2.90GHz clock and four cores; it is a good representation of a common desktop system. It is important to note that the Haswell system does have RAPL energy counters including DRAM counters.

4.1.1.2 Deneb (Phenom)

Deneb has an AMD Phenom II X4 955 with a Deneb architecture. The system is comparable to the Haswell system with a 3.2GHz clock and four cores. This system does *not* have any integrated power or energy counters.

4.1.2 Server Systems

A rack with two high-end server systems was also used. Both Intel and AMD are represented. Great difficulty was experienced in directly measuring power and energy with custom instrumentation as both systems use HP Common Slot Power Supplies. The connection between the power supply and the system is via direct contacts unlike a standard ATX power supply which transmits power via wires. To solve this problem a custom extender board was designed and built. The common slot extender is discussed in detail in Section 3.1.

4.1.2.1 Sandy Bridge

The Intel server machine has two 2.5GHz E5-2640 Sandybridge-EP processors, each with six cores and two threads per core which allows for a total of 24 threads of execution. The system is an HP ProLiant DL360P Gen8 server which features the IPMI interface mentioned earlier in Section 4.1 as well as Intel RAPL counters. The hardware *should* have DRAM RAPL counters available but for some reason they are disabled by the firmware. The IPMI interface is currently off-line on the system; one of the power supplies was damaged during testing and IPMI has not worked since then.

4.1.2.2 Piledriver

The AMD server system is equipped with an Opteron 6376 clock at 2.30GHz. It also has two physical processors which have eight cores each with two threads per core for a total of 32 threads of execution. The AMD machine is an HP ProLiant DL385P Gen8 and also has the aforementioned IPMIinterface as well as APM performance counters.

4.2 Benchmarks

Several benchmarks were used for experimentation in order to stress a range of architectural features. Benchmarks were chosen in order to isolate and test system submodules separately in order to gain information about power and energy usage of those subsystems. The four primary benchmarks are High Performance Linpack (HPL), STREAM, IOzone, and equake which were chosen as each exercises a different part of the system (overall, memory, disk I/O and floating point respectively).

4.2.1 Parallel Libraries

There are a couple things to consider when running parallel benchmarks: which Message Passing Interface (MPI) is going to be used, and in the case of some benchmarks, what Basic Linear Algebra Subprograms (BLAS) package will be used. These decisions should be made on a case-by-case basis as differences in architecture, and interprocess communication can highly influence performance of each interface.

4.2.1.1 Message Passing Interface Implementations

The MPI specification is designed to allow for efficiency and portability. The specification defines how communication occurs between nodes in a distributed system. MPI provides many functions for synchronization, locking, data dissemination, and data aggregation. MPI can take complex data structures and divide the workload between nodes in an intelligent fashion. The main choices for MPI libraries are MPICH (MPICH) [33] and Open Message Passing Interface (Open MPI) [27]. Both interfaces are intended for highly parallel distributed memory applications. MPICH is used in all systems for this research.

4.2.1.2 Basic Linear Algebra Subprograms

BLAS is a specification for highly efficient linear algebra operations. Common operations include vector addition, scalar multiplication, linear combinations, matrix multiplication, and dot products. Most BLAS libraries have C and Fortran interfaces. Some well known BLAS implementations are Automatically Tuned Linear Algebra Software (ATLAS) [76], Netlib Basic Linear Algebra Subprogram (Netlib BLAS) [8], and Open Basic Linear Algebra Subprograms (Open BLAS) [2]. For simplicity, all test systems use ATLAS in an attempt to increase consistency while possibly sacrificing some performance.

4.2.2 High-Performance Linpack

HPL [62] is widely used to benchmark highly parallel computing systems. HPL stresses the floating point capabilities of a system by solving "a (random) dense linear system in double precision (64 bits) arithmetic on distributed-memory computers". HPL's most widely visible contribution is in the TOP500 list of super computers [5] where computing clusters around the world run HPL and submit their results in an attempt to claim a spot on the list. The HPL benchmark is run with the command in Listing 4.2.2. The `-np` flag is used to specify the number of execution threads to be used in the benchmark. The bulk of the configuration is in a text file called `HPL.dat` which controls the problem size, how to split the problem up between nodes, and a large number of other options controlling the algorithm.

```
mpirun −np24 ./xhpl
```

4.2.3 STREAM

The STREAM [55] benchmark was used to test the memory subsystems by loading large amounts of data into memory. STREAM is designed to report an accurate estimate of a system's practical memory bandwidth. STREAM can be used to benchmark both single-core processors as well as distributed systems by using a message passing interface. The STREAM benchmark is run by simply calling the executable as follows ./stream_c or ./stream_omp. The problem size can be modified at compile time by change a value in the Makefile.

4.2.4 IOzone

IOzone [60] tests the filesystem performance by investigating a wide range of filesystem operations including read, write, re-read, re-write, backward read, strided read, fread, fwrite, random read and several others. The command `./iozone -a` runs an automated version of IOzone that adjusts to the system limitations.

4.2.5 Equake

The equake benchmark is an example of a more realistic workload that might be seen in academia or in the scientific community. Equake is a part of the Standard Performance Evaluation Corporation (SPEC) CPU 2000 [71] benchmark suite. Equake simulates "the propagation of elastic waves in large, highly heterogeneous valleys". The result is a highly data-driven benchmark that stresses multiple aspects of a system and is an example of a realistic workload. equake is run simple by calling the executable like and redirecting the input sample into the program as follows `./equake < ./inp.in`.

4.3 Performance Counters

Performance counters will be used where possible in order to gauge performance and energy usage. A mix of hardware and software performance counters will be used on a system-by-system basis. On test systems that have specialized power and energy performance counters they will be compared to experimental instrumentation results. Further information on system specific performance counters is detailed in Section 4.1.

4.4 Experimental Procedure

The experimental procedure requires careful coordination of several hardware components, an embedded system, a modified perf program, and a variety of benchmarks. Several tools including git, mpirun, and bash scripting were used to streamline this process.

4.4.1 Git

Several git repositories were used to gather together information about the instrumentation and perf source code, benchmarks, and resulting data from tests. Git

```
sudo env INTERVAL=1000 IPMI=1 RAPL=1 RAPL_DRAM=0 APM=0 ./
    bench_template.sh ../workload
sudo env INTERVAL=100 IPMI=1 RAPL=0 RAPL_DRAM=0 APM=1 ./
    bench_template.sh ../workload
```

Figure 4.1: Benchmark Template Usages

allows for an easy way to track and distribute changes to individual parts of the system. Any computer with internet access and a user account can easily get access to these repositories.

4.4.2 mpirun

Mpirun is provided by MPICH and is used to dispatch benchmarks and distribute them over the test system nodes. The number of threads and the benchmark must be specified during use. A common usage is similar to `mpirun -np 24 ./xhpl`.

4.4.3 Benchmark Script

A bash script was written in order to automate some of the configuration of the system and the benchmarking tools. With perf, RAPL, APM, and IPMI it can become difficult to know what to execute when running a benchmark. These options can be specified as environmental variables which will be recognized by the script in order to configure test features on the fly. Some usages are shown in Figure 4.1.

The first line is an example of a benchmark run on the Sandy Bridge test system which has IPMI and RAPL features. The second is run on the Piledriver system which has IPMI and APM counters.

The benchmark script is included in Appendix B. The script performs several functions to assist in automating and aggregating data from a variety of sources. Data can come from three major sources: perf, IPMI, and APM. All data gathered from perf is associated automatically with a timestamp but data from IPMI and

APM is simply given as a value with a one time request. The script uses the Linux watch tool in order to periodically request data from IPMI and APM. Watch is also used to generate a timestamp for each sample.

4.4.4 Benchmark Inputs

The test inputs to the benchmarks are included for the sake of completeness. All equake benchmarks were run using the same test input that is omitted due to its size. The input data set is roughly 8MB. The IOzone benchmark uses the `-a` option which automatically adjusts the workload size. Tests with the STREAM benchmark vary in problem size over the architectures to keep runtime at roughly two minutes. Haswellian uses an array size of 100000000, Phenom 75000000, Piledriver 100000000, and Sandy Bridge 100000000. The Phenom array size had to be lowered to 75000000 as the full 100000000 size array exceeded physical memory and cause large decreases in performance. The HPL.dat input files are available in Appendix E

CHAPTER 5
RESULTS AND DISCUSSION

This chapter goes over the experimental results and attempts to compare and validate the custom serial power logging interface with existing integrated and off-the-shelf solutions including IPMI, RAPL, APM, and WUP. The power usage over the courses of a benchmark and the power trends will be compared as well as the total energy usage. The equake benchmark will also be looked at with the energy delay metric in an attempt to gain insight into the power and performance characteristic of the four test machines.

5.1 Results

This section covers experimental results and compares collected serial power and energy readings with built in performance counters and off-the-shelf solutions. Serial power supply readings will be compared with IPMI, APM, WUP, and RAPL values. Serial DRAM values will be compared only to RAPL DRAM values. These comparisons will be made on an architecture-to-architecture basis covering all four test machines. Both the power trend over the course of the benchmark and the total energy consumed will be discussed. Although several instances of each benchmark were run, only a single figure will be shown in each case. Aggregated energy data over all instances will be discussed.

To summarize the major features detailed in Section 4.1: Sandy Bridge has IPMI and RAPL counters, Piledriver has IPMI and APM features, Haswellian has RAPL features including DRAM RAPL counters, and Phenom has no dram or embedded power or energy metrics. All tests are also compared side-by-side with WUP data.

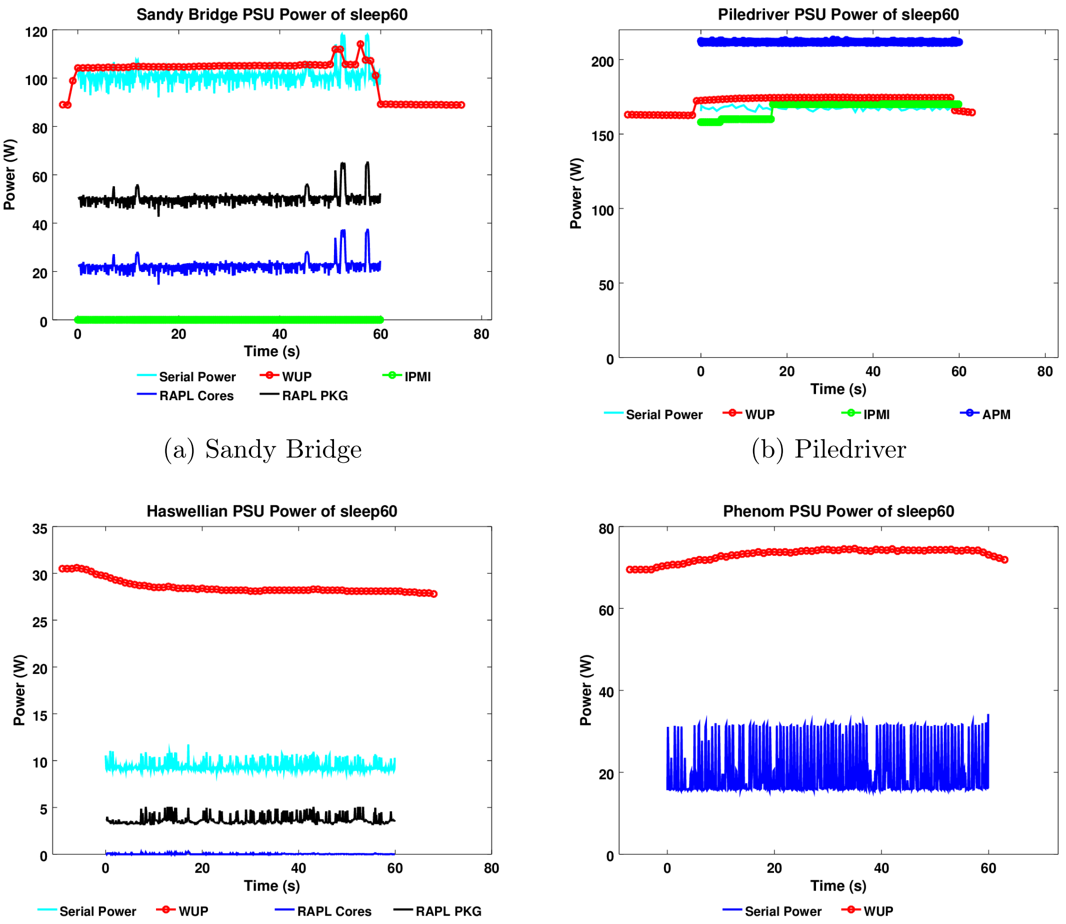
It is important to note that the WUP is measuring the power coming out of the wall outlet whereas the serial system is measuring the power out of the power supply. Power supplies are not 100% efficient so some power is lost in the transition. The CS power supplies claim an efficiency from 89% to 94% under load (with a higher efficiency under higher load).

5.1.1 Sleep

First, the idle power of the system is measured with all available metrics using a simple sleep benchmark. The benchmark is run for 60 seconds and all results are shown in Figure 5.1.

It can be clearly seen that all metrics give a decent picture of the power trends of the idle systems. The sleep benchmark shows what is mostly expected: a constant power reading across all architectures. Some small variations do exist such as in the Sandy Bridge test. This is likely due to background noise or processes as it is only on the order of 10W. These small spikes are seen in all available metrics so it can be assumed to be a variation in system power and not an erroneous deviation in the metrics themselves. The IPMI readings for Piledriver also shows a small increase at 20 seconds into the benchmark. Overall, the sleep benchmark does a reasonable job of establishing a baseline for power usage.

The WUP data for the two server architectures increases slightly while running the sleep benchmark. This is likely due to the overhead from gathering the IPMI data, APM data, and possibly from the serial overhead of the perf hack. The CPU conducting the measurement possibly cannot go into deep sleep states if it is being woken up periodically to do measurements. The relatively stable levels of Haswellian and Phenom seem to indicate that the serial overhead is less of a factor compared to the IPMI and APM polling.



(a) Sandy Bridge

(b) Piledriver

(c) Haswellian

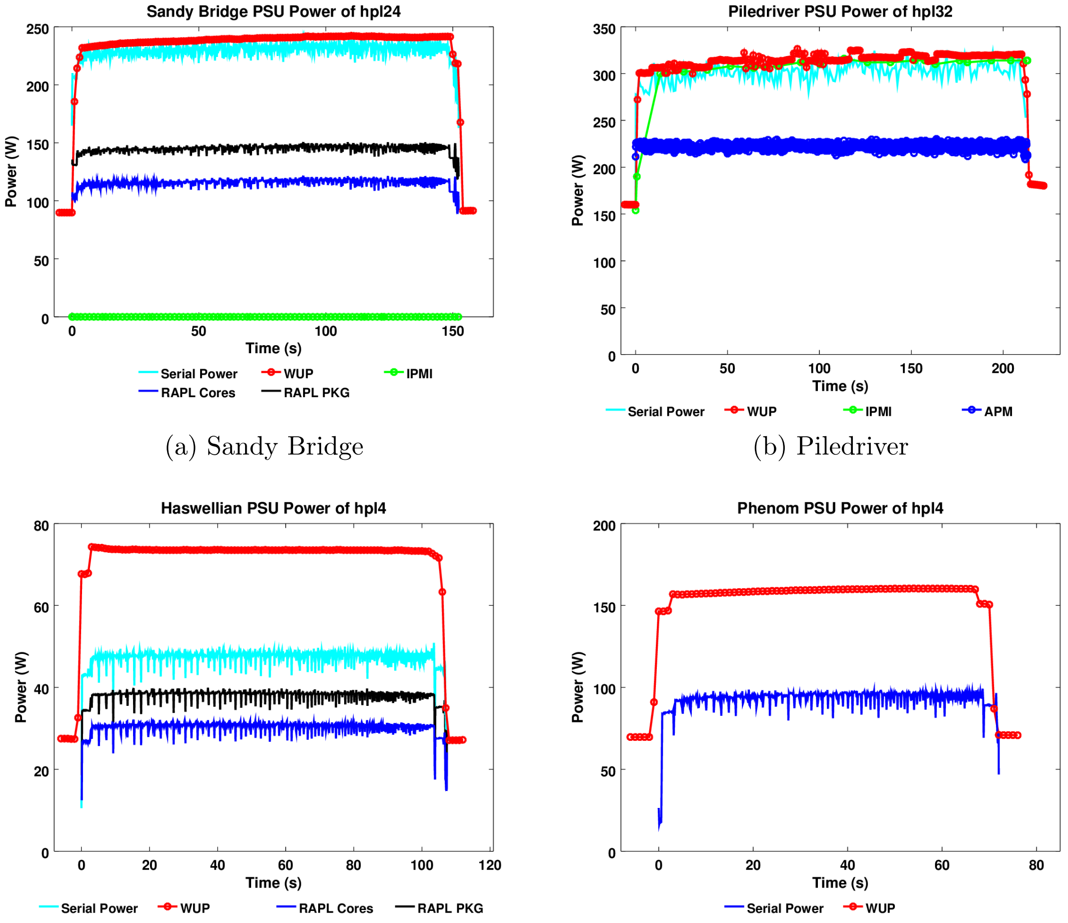
(d) Phenom

Figure 5.1: Sleep on all architectures

5.1.2 High Performance Linpack

Next, HPL is used to stress a variety of subsystems including memory and the floating point modules. Figure 5.2 shows the resulting power trends.

For Sandy Bridge and Piledriver, the serial power can be seen to closely follow the WUP data although slightly low. Regular dips in power may be related to synchronization across the many cores; this needs further investigation. The IPMI readings for Piledriver closely coincide with both the serial readings and the WUP data. For Haswellian and Phenom, no clear claims can be made as only the CPU is being measured by the serial system while the WUP is measuring total system



(a) Sandy Bridge

(b) Piledriver

(c) Haswellian

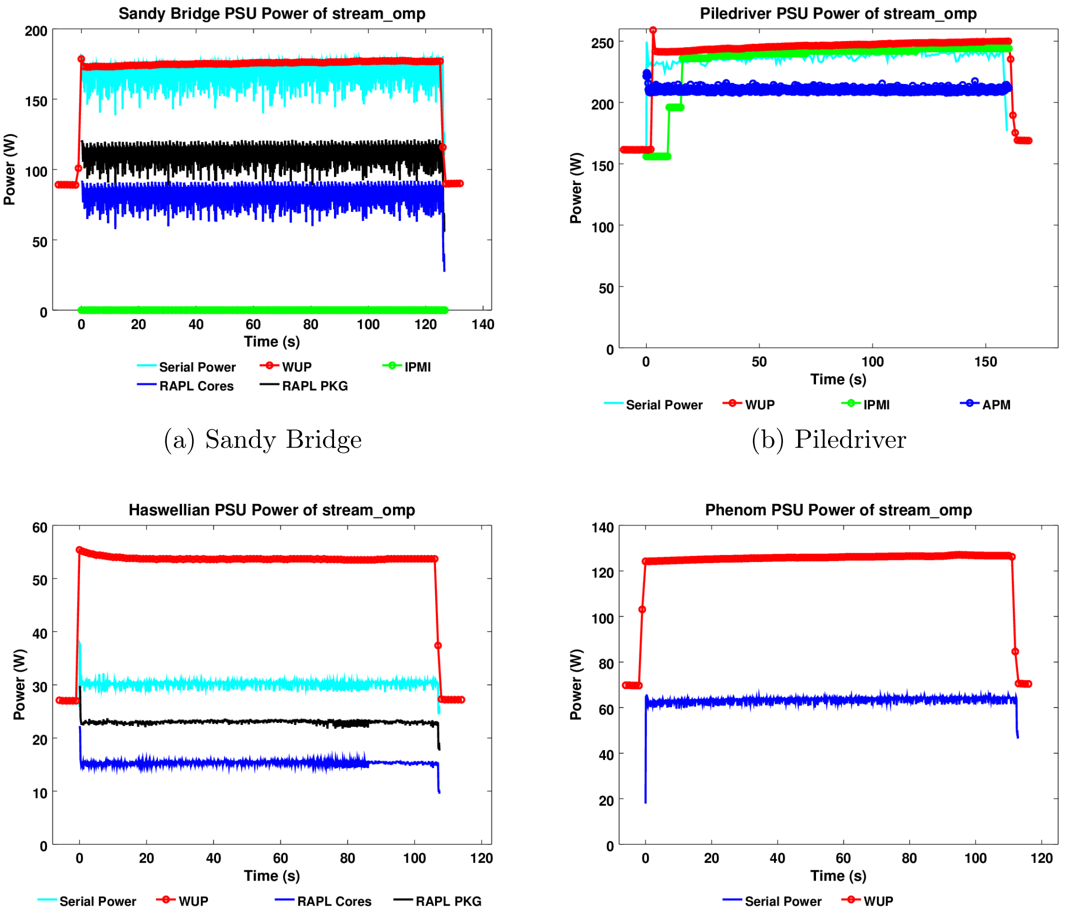
(d) Phenom

Figure 5.2: HPL on all Architectures

power. The RAPL features of Haswellian closely follow the serial CPU power and the many spikes and lulls are seen in the three metrics (though off by a constant factor). Phenom shows that the serial and WUP readings follow the same general trend.

5.1.3 STREAM

Next, the STREAM benchmark is used to stress the memory subsystems. The total system power is much lower than seen from HPL in Figure 5.2 as only the



(a) Sandy Bridge

(b) Piledriver

(c) Haswellian

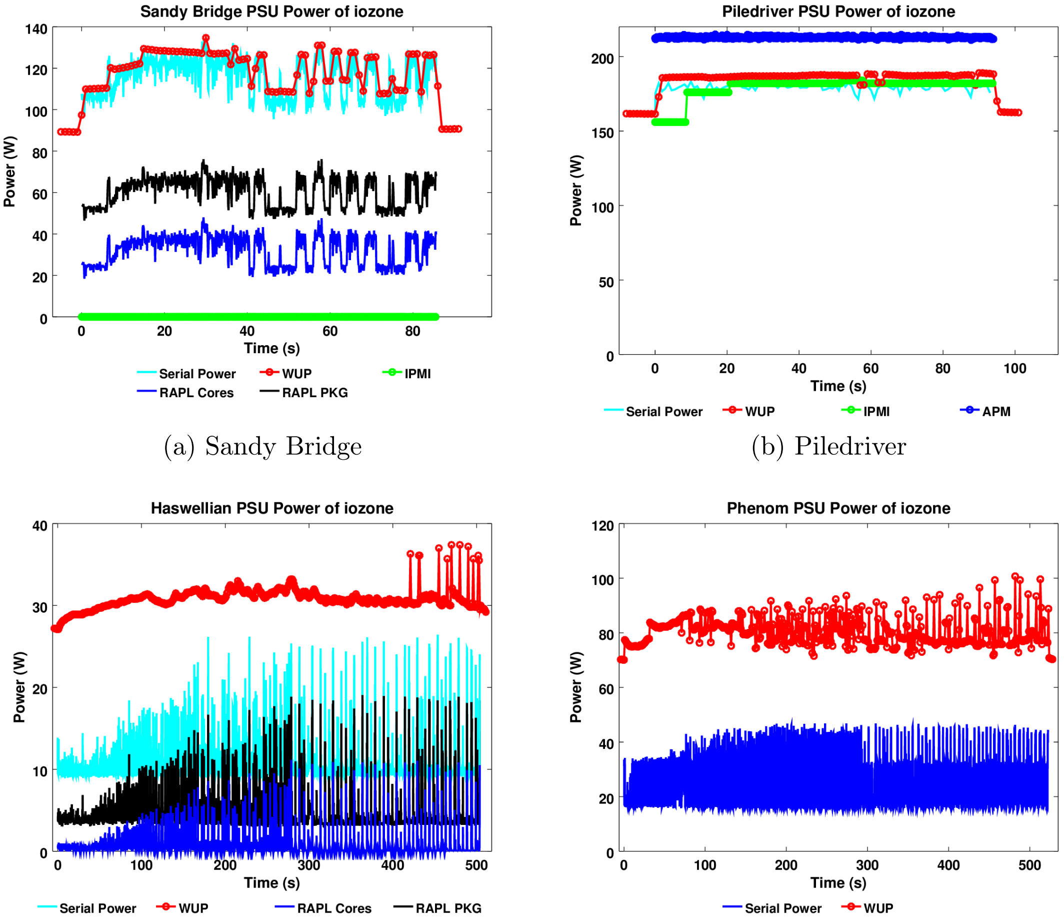
(d) Phenom

Figure 5.3: Parallel STREAM on all Architectures

memory subsystems are being stressed. Figure D.4 gives insight into the behavior of the single-threaded STREAM benchmark.

Unlike the HPL figures, it is very difficult to see the trend in power in these graphs due to the small change in power from the baseline (on the order of 20 to 40 watts). The power trends are generally constant throughout the length of the benchmark which is to be expected from a benchmark of this nature.

Next, the parallel STREAM benchmark is shown in Figure 5.3. The parallel version of STREAM uses Open Multi-Processing (Open MP) to parallelize the workload.



(a) Sandy Bridge

(b) Piledriver

(c) Haswellian

(d) Phenom

Figure 5.4: IOzone on all Architectures

Again, the STREAM results look almost the same as the single-threaded workload. The overall power consumption can be seen to increase compared to their single-threaded counterparts. As before, WUP, IPMI, and serial readings can be seen to follow the same trends.

5.1.4 IOzone

The IOzone benchmark stresses file I/O. Some of the variations seen in these results are unexpected. Figure 5.4 shows the relevant graphs.

All architectures exhibited a large amount of variation during the workload. Sandy Bridge experienced swings in power from roughly 110 watts to 130 watts. Piledriver stayed quite steady at roughly 200 watts while Haswellian and Phenom exhibited large spikes in power when looking at RAPL counters and serial readings (the WUP sampling rate appears be too low to reflect this behavior).

5.1.5 Equake

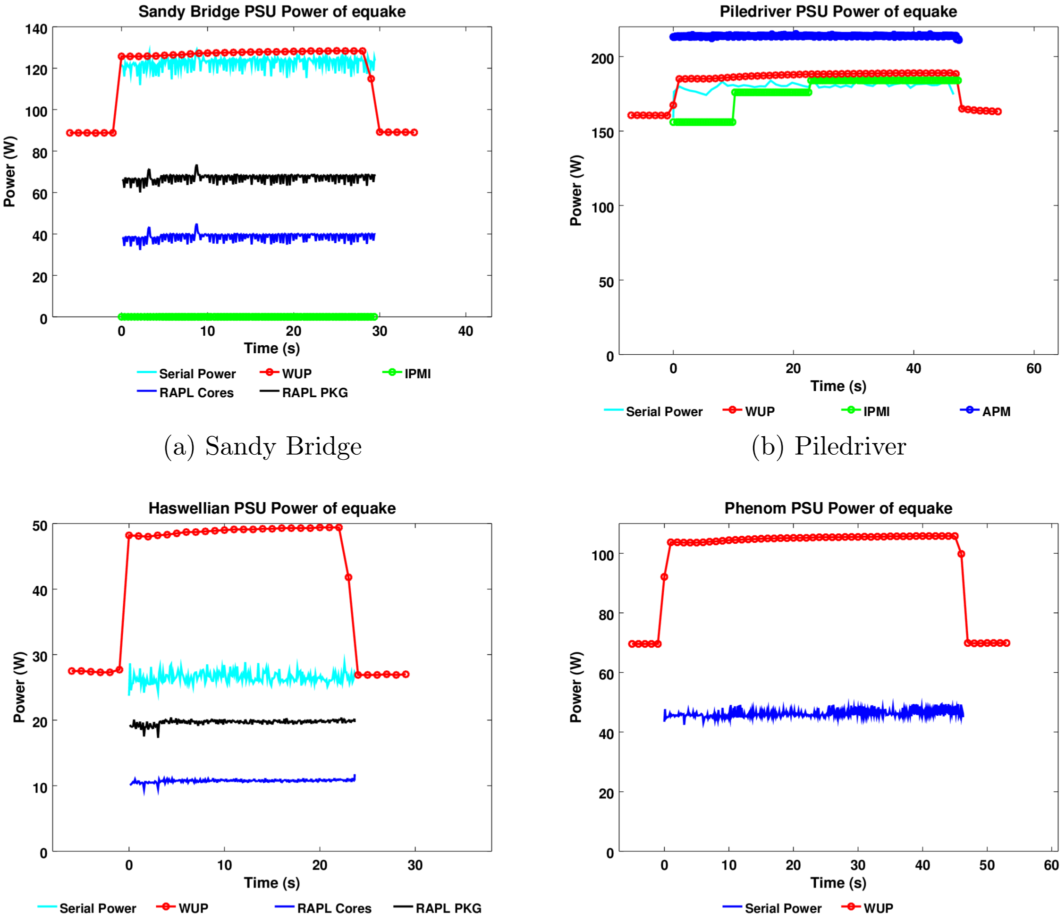
Equake was intended to show results of a realistic benchmark that might be seen in the scientific world. The benchmark failed to stress the system in any significant way. All graphs can be seen in Figure 5.5.

Only Piledriver IPMI power readings show any sort of interesting behavior, increasing by 50 watts at roughly 10 seconds into the benchmark and again by 10 watts at about 25 seconds into the benchmark before leveling out.

Next, the Haswell DRAM RAPL counters will be briefly compared with the serial DRAM measurements. Figure 5.6 shows five plots for the various benchmarks and their serial DRAM measurements and the RAPL DRAM estimations.

5.1.6 DRAM Results

The RAPL values are very clean with little variation that might be expected when switching between multiple threads and the kernel. The serial DRAM measurements on the other hand are quite noisy and some interesting trends can be seen across the benchmarks. Few of these trends are consistently reflected in the RAPL values however which might be due to the fact that the RAPL readings are results of a mathematical model based on memory related instructions. It is possible that the model does not account for some of the finer details in the DRAM module such as refresh.



(a) Sandy Bridge

(b) Piledriver

(c) Haswellian

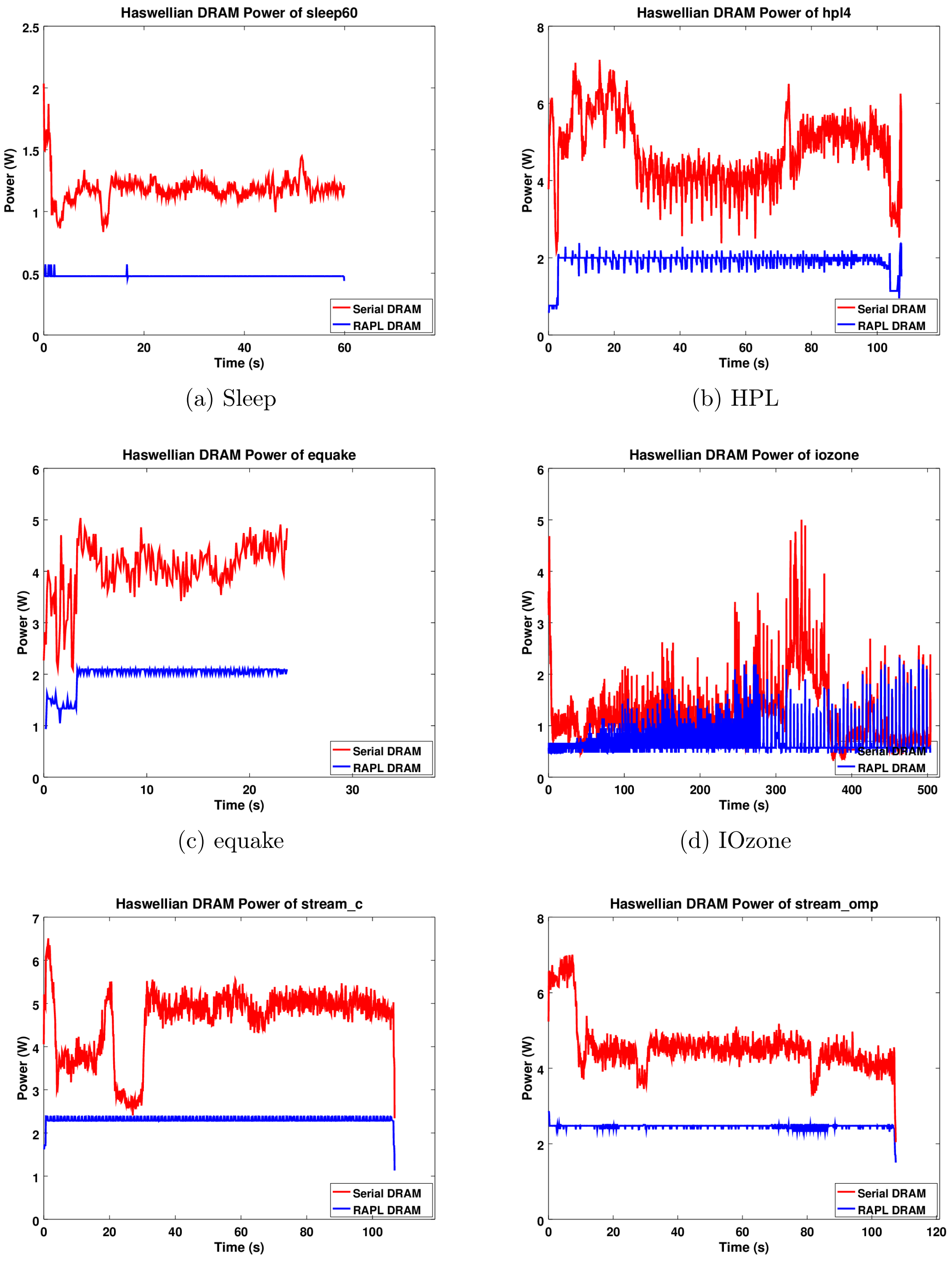
(d) Phenom

Figure 5.5: Equake on all Architectures

Additionally, the RAPL values appear to be consistently low by a factor of two. I am still investigating the source of this error. One possibility is that the published scaling factor is wrong, as until recently the Haswell-EP (server Haswell) RAPL driver had an improper scaling factor.

5.1.7 Energy Results

This section presents several tables of energy usage statistics. All benchmarks and their energy usage behaviors are investigated. Energy delay will be will be looked at for only the equake benchmark. A value of 0.00 means that the metric



(a) Sleep

(b) HPL

(c) equake

(d) IOzone

(e) STREAM

(f) Parallel STREAM

Figure 5.6: Haswell DRAM RAPL results on all Benchmarks

is unavailable for that architecture. It should be noted that due to the manual synchronization of the WUP data and the fact that the range of the WUP data extends both before and after the range of all other data sources that the WUP energy results may be slightly skewed to the high side. Care is taken during data processing to minimize this source of error.

First, the sleep benchmark is used to establish a baseline of energy usage over a 60 second period. Table 5.1 and Table 5.2 show the sleep benchmark results. Comparing the serial system energy values for Piledriver and Sandy Bridge in column three with the WUP data in column four shows a very close relationship between the two. Percent error for serial energy usage for Piledriver only deviates by 0.75% from the WUP values. Sandy Bridge serial energy usage deviates by 3.07%. Comparing the Piledriver serial energy usage with the IPMI values also results in a very small percent error of 1.03%. The Piledriver APM values however deviate wildly and as mentioned in Section 5.2; it is likely that they are non-functional. Nothing significant can be said of DRAM energy usage in three of the tests machines. A brief analysis will be done comparing RAPL DRAM energy values with serial DRAM values in Section 5.1.8.

Comparing the serial and RAPL readings for Sandy Bridge reveal inconclusive results. The serial measurements on the ATX P4 line are measuring total CPU power while the Core and PKG RAPL values are measuring subsets of the CPU functionality. The results support this idea as the serial energy values are always larger than both the RAPL readings.

Table 5.3 and Table 5.4 show the energy usage statistics of the equake benchmarks across all four systems. Again, the energy usage data for the serial system and the WUP correspond quite closely with a percent error of 0.58% for Piledriver and 1.98% for Sandy Bridge. The IPMI has a 1.67% error compared to the WUP. Again, the Haswellian results show that the serial ATX P4 measurements and the RAPL

Table 5.1: PSU and CPU Energy Results for Sleep

Architecture	Delay	PSU	WUP	IPMI	Core	PKG
Haswellian	59.90s	562.07J	1643.10J	0.00J	1.45J	218.27J
Phenom	59.93s	1201.80J	4271.60J	0.00J	0.00J	0.00J
Piledriver	59.11s	9898.47J	10099.50J	10001.84J	0.00J	0.00J
Sandy Bridge	59.90s	6044.02J	6111.50J	0.00J	1355.44J	3027.81J

Table 5.2: DRAM Energy Results for Sleep

Architecture	Delay	Serial DRAM	RAPL DRAM
Haswellian	59.90s	70.76J	28.64J
Phenom	59.93s	55.93J	0.00J
Piledriver	59.11s	178.94J	0.00J
Sandy Bridge	59.90s	54.26J	0.00J

Table 5.3: PSU and CPU Energy Results for equake

Architecture	Delay	PSU	WUP	IPMI	Core	PKG
Haswellian	23.64s	625.62J	1064.35J	0.00J	253.66J	465.55J
Phenom	46.31s	2146.64J	4724.05J	0.00J	0.00J	0.00J
Piledriver	46.59s	8403.20J	8452.80J	8311.60J	0.00J	0.00J
Sandy Bridge	31.58s	3859.35J	3809.30J	0.00J	1219.75J	2119.07J

Table 5.4: DRAM Energy Results for equake

Architecture	Delay	Serial DRAM	RAPL DRAM
Haswellian	23.64s	101.67J	46.57J
Phenom	46.31s	96.58J	0.00J
Piledriver	46.59s	172.80J	0.00J
Sandy Bridge	31.58s	64.31J	0.00J

readings have similar levels of energy consumption. The differences appear to be scaled when comparing them to the sleep benchmark which supports the idea that the RAPL readings are omitting certain modules or functionality when computing the results. RAPL Core differs by 147.03% and PKG differs by 34.41%.

The HPL energy usages figures show similar trends to the previous benchmarks. The serial power supply measurements, IPMI, and WUP. Sandy Bridge has a 2.68% error for the serial power measurements while Piledriver has 3.49% and 0.92% for serial and IPMI respectively. Once more, Haswellian energy results show a clear

Table 5.5: PSU and CPU Energy Results for HPL

Architecture	Delay	PSU	WUP	IPMI	Core	PKG
Haswellian	108.66s	5163.08J	7818.28J	0.00J	3286.84J	4123.00J
Phenom	71.56s	6694.87J	11070.18J	0.00J	0.00J	0.00J
Piledriver	212.65s	64953.20J	67302.84J	66681.55J	0.00J	0.00J
Sandy Bridge	152.31s	35142.29J	36110.10J	0.00J	17717.70J	22158.96J

Table 5.6: DRAM Energy Results for HPL

Architecture	Delay	Serial DRAM	RAPL DRAM
Haswellian	108.66s	447.34J	205.99J
Phenom	71.56s	211.97J	0.00J
Piledriver	212.65s	1474.15J	0.00J
Sandy Bridge	152.31s	560.35J	0.00J

difference between the three metrics. Serial ATX P4 readings differ from RAPL Core readings by 57.77% and from RAPL PKG readings by 25.70%. The difference in percent error between the equake and HPL benchmarks lends support to the fact that the missing energy information is not simply a scale factor or a static overhead or leakage but is a large and dynamic factor in energy consumption within the CPU.

Table 5.7 and Table 5.8 show that Piledriver has 2.30% and 1.91% error for serial and IPMI. Sandy Bridge has 2.37% error for serial results. Haswellian RAPL results are even more skewed than previous benchmarks with 1640.25% error on Core and 128.77% on PKG. These enormous differences may indicate that the RAPL counters are ignoring some factor concerned with disk I/O.

For the STREAM benchmark, Piledriver has a 3.79% error for serial measurements and a 2.45% error for IPMI readings. Sandy Bridge has a 2.59% error for serial energy measurements. Haswellian shows a 182.61% error on RAPL Core readings and a 38.18% error on PKG.

Finally, the parallel STREAM benchmarks reveals the same trend as all the other benchmarks with the serial energy measurements and IPMI measurements being within a few percentage points of the WUP data. Piledriver has a serial

Table 5.7: PSU and CPU Energy Results for IOzone

Architecture	Delay	PSU	WUP	IPMI	Core	PKG
Haswellian	524.74s	5621.91J	16181.20J	0.00J	323.89J	2457.10J
Phenom	521.73s	12707.48J	41775.20J	0.00J	0.00J	0.00J
Piledriver	98.00s	17629.14J	18045.95J	17701.87J	0.00J	0.00J
Sandy Bridge	85.29s	12892.32J	13233.15J	8396.71J	1304.91J	2399.37J

Table 5.8: DRAM Energy Results for IOzone

Architecture	Delay	Serial DRAM	RAPL DRAM
Haswellian	524.74s	554.95J	332.22J
Phenom	521.73s	681.81J	0.00J
Piledriver	98.00s	311.53J	0.00J
Sandy Bridge	85.29s	222.67J	0.00J

Table 5.9: PSU and CPU Energy Results for STREAM

Architecture	Delay	PSU	WUP	IPMI	Core	PKG
Haswellian	106.26s	2439.94J	4841.37J	0.00J	863.25J	1765.64J
Phenom	117.49s	5266.59J	12190.47J	0.00J	0.00J	0.00J
Piledriver	166.99s	30396.62J	31595.37J	30818.35J	0.00J	0.00J
Sandy Bridge	122.88s	15345.50J	15753.70J	0.00J	4859.53J	8387.49J

Table 5.10: DRAM Energy Results for STREAM

Architecture	Delay	Serial DRAM	RAPL DRAM
Haswellian	106.26s	465.04J	246.42J
Phenom	117.49s	347.33J	0.00J
Piledriver	166.99s	762.93J	0.00J
Sandy Bridge	122.88s	392.13J	0.00J

power error of 3.00% and an IPMI error of 2.89% Haswellian again shows large deviations in RAPL and serial readings with 97.8% error on the Core counter and 31.76% on the PKG.

5.1.8 DRAM Energy Results

The tables in Section 5.1.7 show results for both system and CPU energy and DRAM energy usage. DRAM energy statistics can only be compared on the Haswellian test system as it has RAPL DRAM counters and was instrumented with the JET-5464. The results show no clear relationship between the serial measurements and

Table 5.11: PSU and CPU Energy Results for Parallel STREAM

Architecture	Delay	PSU	WUP	IPMI	Core	PKG
Haswellian	107.21s	3240.50J	5670.30J	0.00J	1638.03J	2459.05J
Phenom	112.78s	7157.16J	13937.83J	0.00J	0.00J	0.00J
Piledriver	158.76s	37756.66J	38927.37J	37801.11J	0.00J	0.00J
Sandy Bridge	126.48s	21445.60J	21988.40J	0.00J	10414.41J	14083.71J

Table 5.12: DRAM Energy Results for Parallel STREAM

Architecture	Delay	Serial DRAM	RAPL DRAM
Haswellian	107.21s	465.97J	264.35J
Phenom	112.78s	361.03J	0.00J
Piledriver	158.76s	1022.66J	0.00J
Sandy Bridge	126.48s	581.39J	0.00J

the DRAM performance counters. The serial measurements are scaled by roughly a factor of two compared to the RAPL counters. It is possible that there is a problem with the RAPL counters on the architecture or with the instrumentation itself.

5.1.9 Energy Delay of Equake

Next, the equake benchmark is examined across all architectures with the energy delay metric. Energy delay is used in an attempt to objectively characterize the performance and power behavior of systems over a single workload. The equation for energy delay is giving in Equation 1.4 and is most simply described as an energy measurement with another factor of time added in. The resulting energy delay values are given in Table 5.14 and Table 5.13.

Energy delay values put more emphasis on time instead of just purely on energy as a way of measuring performance. The resulting value is essentially the energy usage scaled by the time it takes each test system to run the benchmark. A slow test machine will have a larger energy delay value than a much faster machine if both machines have the same energy usage. A low energy delay value is desirable. In order of energy delay performance, Haswellian is first with 25157.89Js, then Sandy Bridge with 119975.39Js, followed by Phenom with 218783.34Js, and finally

Table 5.13: PSU and CPU Energy Delay Results for equake

Architecture	Delay (s)	PSU (Js)	WUP (Js)	IPMI (Js)	Core (Js)	PKG (Js)
Haswellian	23.64s	14787.66	25157.89	0.00	5995.61	11004.26
Phenom	46.31s	99416.70	218783.34	0.00	0.00	0.00
Piledriver	46.59s	391534.88	393845.75	392566.67	0.00	0.00
Sandy Bridge	31.58s	122404.46	119975.39	0.00	38675.55	67204.18

Table 5.14: DRAM Energy Delay Results for equake

Architecture	Delay	Serial DRAM	RAPL DRAM
Haswellian	23.64s	2403.22	1100.88
Phenom	46.31s	4472.79	0.00
Piledriver	46.59s	8051.39	0.00
Sandy Bridge	31.58s	2033.18	0.00

Piledriver with 393845.75Js. These values indicate that Haswellian provides the best mix of performance and energy usage for the equake benchmark.

5.2 Discussion

This section will cover the relative effectiveness of each method used to gather power and energy data as discussed in Section 5.1. IPMI, APM, RAPL, and the serial measurement system will be discussed. The perf tool and a brief look at energy delay will also be shown.

5.2.1 Intelligent Plaform Management Interface

IPMI was found to be rather accurate compared to other methods. Although the temporal resolution is low at one sample per second, the power readings can be closely correlated with WUP and serial power data with an average error of 1.79% across all benchmarks. As mentioned before some of this error may be due to incorrect windowing mentioned in Section 5.1.

5.2.2 Application Power Management

APM accuracy is currently unable to be determined, possibly due to some sort of hardware or software bug. The output of the APM module reads as a nearly constant value throughout each test, regardless of load. Further testing and troubleshooting needs to be done in order to determine the usefulness of APM.

5.2.3 Running Average Power Limit

RAPL results appear to be reasonably accurate compared to other methods. It is difficult to use Sandy Bridge results to verify CPU RAPL counters because all other power measurement methods are looking at total system power and not just CPU power. Haswellian results can be used to a degree to show the accuracy of RAPL counters in predicting general trends. The serial values, RAPL Cores, and RAPL PKG can be seen to closely show the same behavior while differing by a varying amount. This large difference in energy usage may be due to an omission of certain CPU functions in the RAPL energy model.

5.2.4 DRAM Measurements

DRAM values appear to be consistently off for RAPL and serial instrumentation. The general power trends seen between the two exist but are not nearly as pronounced as the trends seen in CPU or system power results. The RAPL DRAM values tend to be a fairly consistent factor of two smaller than the serial DRAM readings. It is possible that this is due to a Kernel scaling bug or a problem with the RAPL hardware support. It is also possible that the custom instrumentation is flawed in some way.

5.2.5 watts up? PRO

As the only off-the-shelf solution the WUP claims a 1.5% accuracy. The WUP results are used as a reference although the low sampling rate makes fine details hard to see.

5.2.6 Serial Instrumentation

The serial instrumentation can be seen to closely follow trends seen in the other methods. Measuring both the CS supplies and ATX connectors reveals results that very closely follow the WUP, RAPL values, and IPMI readings. The serial results are off by a small constant factor to the WUP data in cases measuring the CS supplies. This factor may be attributed to the power supply efficiency which ranges from 89% to 94%. Measuring the ATX supplies shows that the serial results are off by a slightly larger factor (as much as 20%). This large deviation is expected though as the ATX P4 connector transmits power only to the CPU and not to the rest of the system whereas the WUP is measuring total system power.

5.2.7 Perf

An unfortunate side-effect of using perf as the tool to sample data is that the events closely surrounding the start and end of the system cannot be measured. It can be difficult to see the increase in power as a result of a benchmark as only the trends during the benchmark can be seen. The only helpful reference is the WUP data gathered.

5.2.8 Equake Energy Delay

The equake energy delay results from Section 5.2.8 describe which test systems performs the best as far as energy and execution time are concerned. The results reveal that Haswellian performs best, followed by Sandy Bridge, Phenom, and

Piledriver. These results have little effectiveness in determining which system performs best in the general case as the equake benchmark does not appear to put a large amount of stress on the various parts of the system. The energy delay metric only indicates which machine performed the best for this set of benchmarks.

CHAPTER 6
CONCLUSION AND FUTURE WORK

This section summarizes the findings regarding the integrated power and energy metrics, the serial power measurement system, and off-the-shelf solutions. Problems and potential improvements to the serial power measurement system, instrumentation, and accompanying real-time perf interface will be discussed as well as possible future applications of the work.

6.1 Conclusion

The custom instrumentation, embedded system, and perf interface tentatively provide an accurate and precise alternative to current methods of gathering power and energy data such as IPMI, APM, WUP, and RAPL counters. However, more data is needed before any claims can be made to accuracy of the system's Common Slot power supply, ATX P4 or DRAM measurement capabilities.

The custom instrumentation measures high power and low power lines with moderate accuracy and has been tested with as small as a 100ms sampling period. The high power measurements have been tested on CS and ATX power supplies. CS supplies reveal that serial instrumentation has very accurate energy readings with error as large as 3%. ATX P4 power supply lines are difficult to verify as the WUP measures system power and the P4 connector exposes CPU power. It is difficult to validate DRAM measurements as the only alternative metric, RAPL, appears to be in some way flawed on the test machine used. It is also possible that the error lies in our test setup; these possibilities should be investigated further.

The embedded system successfully samples and transmits all necessary voltages in real-time. The perf interface requests and receives the real-time data and associates it with the necessary timestamp before printing to the user. Trends over time in the power data from the various metrics show that synchronization of the various sources of data is sound. Even though the several sources of power and energy data have different timestamps, the benchmarking script manages to bring together all metrics and align them with respect to time. Currently, the overhead and possible impact of the added serial calls to the perf code on a variety of benchmarks is unknown.

6.2 Problems and Improvements

A variety of problems were encountered during the design, implementation, and experimentation with the serial power measurement system. These problems range from the serial protocol, to the perf interface, to problems with the integrated metrics. All problems and possible solutions will be discussed.

6.2.1 Instrumentation

Although the high power instrumentation appears to measure power quite accurately, the low power instrumentation has unverified results. Inconsistencies between the integrated RAPL DRAM readings and the serial DRAM measurements mean that neither are verified with any accuracy. More work should be done to troubleshoot Kernel drivers for the Haswell machine. The low power instrumentation should also be verified using other means such as a logging oscilloscope.

Another possible problem might arise if the Hall effect sensors are maxed out. If the Hall effect sensors are used to measure the maximum of 30A the output voltage will be above the 3.3V ADC reference of the Teensy. This is easily solved by stepping down the output of the Hall effect sensor with an amplifier or a simple resistive voltage divider.

6.2.2 Embedded System

Section 5.1 shows that the embedded system functions quite well in showing trends in power usage as well as being accurate in showing overall system power. Some improvements can be made to the embedded system including: debugging some non-functional ADC channels and improving the serial protocol.

During testing, some ADC lines were found to not respond as expected to test voltages. It is unknown at this time whether this is a configuration problem or a result of hardware failure.

Currently, the embedded system only samples values when instructed to do so by the perf client. An improvement over this would be to have the embedded system continuously sample in the background and provide samples when instructed. This would allow for non-blocking spontaneous requests compared to the current system which must be queried, sample for the desired period, and then transmit results. The improved system would be able to immediately provide serial power data upon request for the previous time interval

6.2.3 Perf

Many problems arose from the perf interface and source code. The current implementation uses a hack to insert calls into the perf stat routine. A massive improvement would be to implement the serial calls as an actual perf event which would allow the usage of other perf tools including source code analysis with perf record.

A memory leak also exists in the current implementation. Due to the possible improvements in the perf interface it may not be practical to hunt down and solve the leak as the implementation is likely to change completely in the future.

6.2.4 Integrated Metrics

When functional, the integrated counters and interfaces provide reasonably accurate results. Due to implementation bugs or hardware failure, these integrated features fail to provide useful result on some occasions. APM was found to be unreliable and only output a power reading that fluctuated around a constant value.

A hardware failure also caused the IPMI interface on Sandy Bridge to produce constant 0.0W at all times. The failure is a result of possible shorting the supply in one of the CS supply slots. The slot is now non-functional and the IPMI interface ceased to function soon after. The second power supply slot still functions and the system continues to run. A blinking orange light on the front of the server indicates "Flashing amber = System health is degraded". The system cannot be powered from the outer CS supply slot anymore and the inner one must be used. These bugs and failures should be investigated and solved in order to move forward and work towards validating their accuracy.

6.2.5 Haswellian Running Average Power Limit DRAM

Initial data seems to indicate that the RAPL DRAM counters for the Haswellian machine are off by roughly a factor of two. This may be a Linux Kernel bug or a hardware problem. It is also a possible problem with the serial low power instrumentation. Further steps should be taken to analyze the current state of Haswell DRAM RAPL counters and the low power instrumentation so any potential bugs or problems might be found.

6.3 Future Work

There are various pieces of future work related to this research. First will be to get the bottom of the various discrepancies in measurements, including the APM issue and the RAPL DRAM issue. More detailed results should be taken with complete benchmark suites, such as SPEC CPU 2006 [72] or PARSEC [11]. More complete system instrumentation needs to be done, including hard-drive, GPU, and fans, in order to determine where all energy in a system is going. Finally, the infrastructure can be used for power and energy optimization of programs. With the instrumented perf utility it should be possible to gather insights into program power behavior that were not possible before.

REFERENCES

[1] KiCad EDA software suite website. http://www.kicad-pcb.org.

[2] OpenBLAS an optimized BLAS library website. http://www.openblas.net/.

[3] Teensy USB development board website. https://www.pjrc.com/teensy/.

[4] Top green 500 list:: Environmentally responsible supercomputing. http://www.green500.org/.

[5] Top 500 supercomputing sites. http://www.top500.org/, 2014.

[6] P. Alonso, R. Badia, J. Labarta, M. Barreda, M. Dolz, R. Mayo, E. Quintana-Ortí, and R. Reyes. Tools for power-energy modelling and analysis of parallel scientific applications. In *In Proc. 41st International Conference on Parallel Processing*, pages 420–429, 2012.

[7] AMD. *AMD Family 15h Processor BIOS and Kernel Developer Guide*, 2011.

[8] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide.* Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999.

[9] R. Aroca and L. Gonçalves. Towards green data centers: A comparison of x86 and ARM architectures power efficiency. *Journal of Parallel and Distributed Computing*, 72:1770–1780, 2012.

[10] D. Bedard, R. Fowler, M. Linn, and A. Porterfield. PowerMon 2: Fine-grained, integrated power measurement. Technical Report TR-09-04, Renaissance Computing Institute, 2009.

[11] C. Bienia, S. Kumar, J. Singh, and K. Li. The PARSEC benchmark suite: Characterization and architectural implications. In *Proc. IEEE/ACM International Conference on Parallel Architectures and Compilation Techniques*, pages 72–81, Oct. 2008.

[12] W. Bircher and L. John. Complete system power estimation: A trickle-down approach based on performance events. In *Proc. IEEE International Symposium on Performance Analysis of Systems and Software*, pages 158–168, Apr. 2007.

[13] S. Browne, J. Dongarra, N. Garner, G. Ho, and P. Mucci. A portable programming interface for performance evaluation on modern processors. *International Journal of High Performance Computing Applications*, 14(3):189–204, 2000.

[14] K. Cameron. Energy oddities part 1: Why the energy world is odd. *Computer*, 46(1):83–84, 2013.

[15] H. Chen, S. Wang, and W. Shi. Where does the power go in a computer system: Experimental analysis and implications. In *International Green Computing Conference*, pages 1–6, July 2011.

[16] M. Cloutier, C. Paradis, and V. Weaver. Design and analysis of a 32-bit embedded high-performance cluster optimized for energy and performance. In *First International Workshop on Hardware-Software Co-Design for High Performance Computing*, Nov. 2014.

[17] G. Contreras and M. Martonosi. Power prediction for Intel XScale processors using performance monitoring unit events. In *Proc. IEEE/ACM International Symposium on Low Power Electronics and Design*, pages 221–226, Aug. 2005.

[18] Z. Cui, Y. Zhu, Y. Bao, and M. Chen. A fine-grained component-level power measurement method. In *International Green Computing Conference*, July 2011.

[19] H. David, E. Gorbatov, U. Hanebutte, R. Khanna, and C. Le. RAPL: Memory power estimation and capping. In *ACM/IEEE International Symposium on Low-Power Electronics and Design*, Aug. 2010.

[20] M. Diouri, M. Dolz, O. Glück, L. Lefèvre, P. Alonso, S. Catalán, R. Mayo, and E. Quintana-Ort. Solving some mysteries in power monitoring of servers: Take care of your wattmeters! In *Energy Efficiency in Large Scale Distributed Systems conference*, Apr. 2013.

[21] J. Dongarra, H. Ltaief, P. Luszczek, and V. Weaver. Energy footprint of advanced dense numerical linear algebra using tile algorithms on multicore architecture. In *Proc. of the 2nd International Conference on Cloud and Green Computing*, Nov. 2012.

[22] D. Economou, S. Rivoire, C. Kozyrakis, and P. Ranganathan. Full-system power analysis and modeling for server environments. In *Proc. Workshop on Modeling, Benchmarking, and Simulation*, June 2006.

[23] Electronic Educational Devices. Watts Up PRO. `http://www.wattsupmeters.com/`, May 2009.

[24] X. Feng, R. Ge, and K. Cameron. Power and energy profiling of scientific applications on distributed systems. In *Proc. 19th IEEE International Parallel and Distributed Processing Sumposium*, page 34, Apr. 2005.

[25] J. Flinn and M. Satyanarayanan. PowerScope: a tool for profiling the energy usage of mobile applications. In *Proc. of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, pages 2–10, Feb. 1999.

[26] M. Floyd, M. Allen-Ware, K. R. K, B. Brock, C. Lefurgy, A. Drake, L. Pesantez, T. Gloekler, J. T. J, P. Bose, and A. Buyuktosunoglu. Introducing the adaptive energy management features of the power7 chip. *IEEE Micro*, 31(2):60–75, 2011.

[27] E. Gabriel, G. Fagg, G. Bosilca, T. Angskun, J. Dongarra, J. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. Castain, D. Daniel, R. Graham, and T. Woodall. Open MPI: Goals, concept, and design of a next generation MPI implementation. In *In Proc. 11th European PVM/MPI Users' Group Meeting*, Sept. 2004.

[28] R. Ge, X. Feng, S. Song, H.-C. Chang, D. Li, and K. Cameron. PowerPack: Energy profiling and analysis of high-performance systems and applications. *IEEE Transactions on Parallel and Distributed Systems*, 21(6), May 2010.

[29] T. Gleixner and I. Molnar. Performance counters for Linux, 2009.

[30] B. Goel, S. McKee, R. Gioiosa, K. Singh, M. Bhadauria, and M. Cesati. Portable, scalable, per-core power estimation for intelligent resource management. In *First International Green Computing Conference*, Aug. 2010.

[31] M. Gottscho, A. Kagalwalla, and P. Gupta. Analyzing power variability of ddr3 dual inline memory modules. Technical report, University of California, Los Angeles, 2011.

[32] M. Gottscho, A. Kagalwalla, and P. Gupta. Power variability in contemporary DRAMs. *IEEE Embedded Systems Letters*, 4(2):37–40, 2012.

[33] W. Gropp. MPICH2: A new start for MPI implementations. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, page 7, Sept. 2002.

[34] D. Hackenberg, T. Ilsche, R. Schoene, D. Molka, M. Schmidt, and W. E. Nagel. Power measurement techniques on standard compute nodes: A quantitative comparison. In *Proc. IEEE International Symposium on Performance Analysis of Systems and Software*, Apr. 2013.

[35] H. Hamann, J. Lacey, A. Weger, and J. Wakil. Spatially-resolved imaging of microprocessor power SIMP: hotspots in microprocessors. In *In Proc. 10th Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronics Systems*, pages 121–125, May 2006.

[36] M. Hennecke, W. Frings, W. Homberg, A. Zitz, M. Knobloch, and H. Böttiger. Measuring power consumption of IBM Blue Gene/P. *Computer Science – Research and Development*, 27:329–336, Nov. 2012.

[37] M. Horowitz, T. Indermaur, and R. Gonzalez. Low-power digital design. In *Proc. of the IEEE Symposium on Low Power Electronics*, pages 8–11, Oct. 1994.

[38] C.-H. Hsu and S. Poole. Power measurement for high performance computing: State of the art. In *International Green Computing Conference*, July 2011.

[39] A. Hylick, R. Sohan, A. Rice, and B. Jones. An analysis of hard drive energy consumption. In *Proc. IEEE 16th International Symposium on Modelling Analysis and Simulation of Computer and Telecommunication Systems*, pages 1–10, Sept. 2008.

[40] Intel. Voltage regulator-down (vrd) 11.0 processor power delivery design guidelines for desktop lga775 socket. http://www.intel.com/content/dam/doc/design-guide/voltage-regulator-down-11-0-processor-power-delivery-guide.pdf, Nov. 2006.

[41] Intel. *Intel Energy Checker: Software Developer Kit User Guide*, 2010.

[42] Intel Corporation. *Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3: System Programming Guide*, Feb. 2014.

[43] M. Jarus, S. Varette, A. Oleksiak, and P. Bouvry. Performance evaluation and energy efficiency of high-density HPC platforms based on Intel, AMD and ARM processors. *Energy Efficiency in Large Scale Distributed Systems*, pages 182–200, 2013.

[44] R. Joseph and M. Martonosi. Run-time power estimation in high-performance microprocessors. In *Proc. IEEE/ACM International Symposium on Low Power Electronics and Design*, pages 135–140, Aug. 2001.

[45] K. Khan, F. Nybäck, Z. Ou, J. Nurminen, T. Niemi, G. Eulisse, P. Elmer, and D. Abdurachmanov. Energy profiling using IgProf. In *CCGrid Poster Session*, May 2015.

[46] S. Khoshbakht and N. Dimopoulos. Relating application memory activity to processor power. In *Proc. International Conference on Parallel Processing*, pages 849–857, Oct. 2013.

[47] R. Knapp, K. Karavanic, and A. Márquez. Integrating power and cooling data into parallel performane analysis. In *Proc. of the 2nd International Workshop on Green Computing*, Sept. 2010.

[48] J. Laros III, D. DeBonis, R. Grant, S. Kelly, M. Levenhagen, S. Olivier, and K. Pedretti. High performance computing – power application programming interface specification version 1.0. Technical Report SAND2014-17061, Sandia National Laboratories, 2014.

[49] J. Laros III, K. Pedretti, S. Kelly, J. Vandyke, C. Vaughan, and M. Swan. Investigating real power usage on red storm. In *Cray Users Group Meeting*, 2009.

[50] M. Laurenzano, A. Tiwari, A. Jundt, J. Peraza, W. Ward Jr., R. Campbell, and L. Carrington. Characterizing the performance-energy tradeoff of small ARM cores in HPC computation. In *Proc. of Euro-Par 2014*, pages 124–137, Aug. 2014.

[51] B. Lee and D. Brooks. Accurate and efficient regression modeling for microarchitectural performance and power prediction. In *Proc. 12th ACM Symposium on Architectural Support for Programming Languages and Operating Systems*, pages 185–194, Oct. 2006.

[52] D. Lee, M. O'Sullivan, and C. Walker. A framework for measuring the performance and power consumption of storage components under typical workload. *GSTF Journal on Computing*, 1(2), Feb. 2011.

[53] A. Mahesri and V. Vardhan. Power consumption breakdown on a modern laptop. In *Proc. of the 4th International Workshop on Power-Aware Computer Systems*, pages 165–180, Dec. 2004.

[54] A. Martin, M. Nyström, and P. Pénzes. ET2: A metric for time and energy efficiency of computation. In *Power Aware Computing*, pages 293–315, 2002.

[55] J. McCalpin. STREAM: Sustainable memory bandwidth in high performance computers. `http://www.cs.virginia.edu/stream/`, 1999.

[56] F. Mesa-Martinez, J. Nayfach-Battilana, and J. Renau. Power model validation through thermal measurements. In *Proc. 34th IEEE/ACM International Symposium on Computer Architecture*, pages 302–311, June 2007.

[57] F. Moghaddam, T. Geenen, P. Lago, and P. Grosso. A user perspective on energy profiling tools in large scale computing environments. In *Sustainable Internet and ICT for Sustainability*, pages 1–5, Apr. 2015.

[58] D. Molka, D. Hackenberg, R. Schone, and M. Muller. Characterizing the energy consumption of data transfers and arithmetic operations on x86-64 processors. In *Proc. International Green Computing Conference*, pages 123–133, Aug. 2010.

[59] M. C. no, S. Catalán, R. Mayo, and E. Quintana-Ortí. Reducing the cost of power monitoring with DC Wattmeters. *Computer Science – Research and Development*, 30(2):107–114, May 2015.

[60] W. Norcott and D. Capps. IOzone file system benchmark. `http://www.iozone.org`.

[61] NVIDIA. *NVML Reference Manual*, 2012.

[62] A. Petitet, R. Whaley, J. Dongarra, and A. Cleary. HPL — a portable implementation of the high-performance linpack benchmark for distributed-memory computers. Innovative Computing Laboratory, Computer Science Department,

University of Tennessee, v2.0, http://www.netlib.org/benchmark/hpl/, Jan. 2008.

[63] L. Piga, R. Bergamasci, R. Azevedo, and S. Rigo. Power measuring infrastructure for computing systems. Technical Report IC-11-09, Universidade Estadual de Campinas, Mar. 2011.

[64] C. Poirier, R. McGowen, C. Bostak, and S. Naffziger. Power and temperature control on a 90nm Itanium ®-family processor. In *Proc. IEEE International Solid State Circuits Conference*, pages 304–305, 2005.

[65] A. Rahmati, M. Hicks, D. Holcomb, and K. Fu. Refreshing thoughts on DRAM: Power saving vs. data integrity. In *Workshop on Approximate Computing Across the System Stack*, Mar. 2014.

[66] E. Rotem, A. Naveh, D. Rajwan, A. Anathakrishnan, and E. Weissmann. Power-management architecture of the Intel microarchitecture code-named Sandy Bridge. *IEEE Micro*, 32(2):20–27, 2012.

[67] D. Schmidt and N. Wehn. DRAM power management and energy consumption: a critical assessment. In *Proc. of the 22nd Annual Symposium on Integrated Circuits and System Design*, Aug. 2009.

[68] D. Singh and W. Kaiser. Energy efficient network data transport through adaptive compression using the DEEP platforms. In *IEEE 8th International Conference on Wireless and Mobile Computing*, pages 253–260, Oct. 2012.

[69] D. Singh, P. Peterson, P. Reiher, and W. Kaiser. The atom LEAP platform for energy-efficient embedded computing: Architecture, operation, and system implementation. Technical report, University of California, Los Angeles, Dec. 2010.

[70] K. Singh, M. Bhadauria, and S. McKee. Real time power estimation and thread scheduling via performance counters. *Computer Architecture News*, 37(2):46–35, July 2009.

[71] Standard Performance Evaluation Corporation. SPEC CPU benchmark suite. http://www.specbench.org/osg/cpu2000/, 2000.

[72] Standard Performance Evaluation Corporation. SPEC CPU benchmark suite. http://www.specbench.org/osg/cpu2006/, 2006.

[73] P. Stanley-Marbell and V. Cabezas. Performance, power, and thermal analysis of low-power processors for scale-out systems. In *Proc of IEEE International Symposium on Parallel and Distributed Processing*, pages 863–870, May 2011.

[74] S. Wallace, V. Vishwanath, S. Coghlan, Z. Lan, and M. Papka. Measuring power consumption on IBM Blue Gene/Q. In *Proc. 9th Workshop on High-Performance Power-Aware Computing*, pages 853–859, May 2013.

[75] V. Weaver, M. Johnson, K. Kasichayanula, J. Ralph, P. Luszczek, D. Terpstra, and S. Moore. Measuring energy and power with papi. In *Proc. of The 1st International Workshop on Power-Aware Systems and Architectures*, Sept. 2012.

[76] R. C. Whaley and J. Dongarra. Automatically tuned linear algebra software. In *Proc. of Ninth SIAM Conference on Parallel Processing for Scientific Computing*, 1999.

[77] W. Wu, L. Jin, and J. Yang. A systematic method for functional unit power estimation in microprocessors. In *Proc. 43rd ACM/IEEE Design Automation Conference*, pages 554–557, July 2006.

[78] J. Yan, C. Lonappan, A. Vajid, D. Singh, and W. Kaiser. Accurate and low-overhead process-level energy estimation for modern hard disk drives. In *IEEE International Conference on Green Computing and Communications*, pages 171–178, Aug. 2013.

[79] K. Yoshii, K. Iskra, R. Gupta, P. Beckman, V. Vishwanath, C. Yu, and S. Coghlan. Evaluating power monitoring capabilites on IBM Blue Gene/P and Blue Gene/Q. In *IEEE International Conference on Cluster Computing*, pages 36–44, Sept. 2012.

Appendix A

PERF TOOL MODIFICATIONS

```
*** builtin-stat.c.bak  2015-05-20 16:03:20.831743610 -0400
--- builtin-stat.c  2015-06-02 12:07:32.227682491 -0400
***************
*** 63,68 ****
--- 63,71 ----
  #include <sys/prctl.h>
  #include <locale.h>

+ /* Added */
+ #include "ssplib.c"
+
  #define DEFAULT_SEPARATOR "␣"
  #define CNTR_NOT_SUPPORTED    "<not␣supported>"
  #define CNTR_NOT_COUNTED  "<not␣counted>"
*************** static void print_interval(void)
*** 500,505 ****
--- 503,515 ----
            print_counter_aggr(counter, prefix);
    }

+   /* Added serial requests */
+   receive_samples();
+   for (int i = 0; i < bit_count(my_channels); i++) {
+       fprintf(output, "%s␣␣␣␣␣%d␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣power␣(W)\n",
   prefix, my_samples[i]);
+   }
+   begin_sample();
+
    fflush(output);
  }

*************** static void handle_initial_delay(void)
*** 515,520 ****
--- 525,533 ----
        evlist__for_each(evsel_list, counter)
            perf_evsel__enable(counter, ncpus, nthreads);
    }
+
+   /* Added serial start polling */
+   begin_sample();
  }

  static volatile int workload_exec_errno;
*************** static int __run_perf_stat(int argc, con
*** 543,548 ****
```

```
--- 556,575 ----
    if (interval) {
        ts.tv_sec  = interval / 1000;
        ts.tv_nsec = (interval % 1000) * 1000000;
+
+       /* Added code for serial init */
+       my_tty_fd = open(my_device, O_RDWR | O_NOCTTY | O_NDELAY);
+       if(my_tty_fd == -1) {
+           fprintf(output, "failed␣to␣open␣port\n");
+
+           return 1;
+       }
+       serial_init();
+       cfsetospeed(&my_tio,B115200);
+       cfsetispeed(&my_tio,B115200);
+       tcsetattr(my_tty_fd, TCSANOW, &my_tio);
+       send_interval(interval);
+       send_channels();
    } else {
        ts.tv_sec  = 1;
        ts.tv_nsec = 0;
*************** static int __run_perf_stat(int argc, con
*** 619,624 ****
--- 646,654 ----
        if (workload_exec_errno) {
            const char *emsg = strerror_r(workload_exec_errno, msg,
                sizeof(msg));
            pr_err("Workload␣failed:␣%s\n", emsg);
+           if (my_tty_fd != -1) {
+               close(my_tty_fd);
+           }
            return -1;
        }

*************** static int __run_perf_stat(int argc, con
*** 650,655 ****
--- 680,690 ----
        }
    }

+
+   /* Added */
+   if (my_tty_fd != -1) {
+       close(my_tty_fd);
+   }
    return WEXITSTATUS(status);
  }
```

Appendix B

BENCHMARK SCRIPT

```
#!/usr/bin/env bash

echo "Run␣this␣script␣like␣the␣following:"
echo "sudo␣env␣OUT_FIL=hplnp2␣INTERVAL=1000␣IPMI=1␣RAPL=1␣RAPL_DRAM
    =1␣APM=0␣./bench_template.sh␣../workload"

: ${OUT_FIL:?"Please␣specify␣the␣output␣filename"}
: ${INTERVAL:?"Please␣specify␣an␣interval"}
: ${IPMI:?"Please␣specify␣IPMI␣support"}
: ${RAPL:?"Please␣specify␣RAPL␣support"}
: ${RAPL_DRAM:?"Please␣specify␣RAPL_DRAM␣support"}
: ${APM:?"Please␣specify␣APM␣support"}

if [ "$#" -eq 0 ]; then
    echo "Specify␣a␣workload..."
    exit 1
fi

if [ "$IPMI" -eq 1 ]; then
    echo "Insert␣ipmi␣kernel␣modules"
    /sbin/modprobe ipmi_msghandler
    /sbin/modprobe ipmi_devintf
    /sbin/modprobe ipmi_si

    IPMI_CMD="ipmitool␣-c␣sensor␣get␣'Power␣Meter'"
fi

if [ "$RAPL" -eq 1 ]; then
    RAPL_CNTRS="-a␣-e␣power/energy-cores/␣\
␣␣␣␣-e␣power/energy-pkg/"
fi

if [ "$RAPL_DRAM" -eq 1 ]; then
    RAPL_CNTRS+="␣-e␣power/energy-ram/"
fi

if [ "$APM" -eq 1 ]; then
    APM_CMD="sensors␣-u"
fi

PERF="../../linux-3.18.13/tools/perf/perf"
PERF_OPTS="stat␣-I$INTERVAL"
# -a is needed for some perf counters
# also need to run this as root or change paranoid values
# Error:
```

```
# You may not have permission to collect system-wide stats.
# Consider tweaking /proc/sys/kernel/perf_event_paranoid:
# -1 - Not paranoid at all
# 0 - Disallow raw tracepoint access for unpriv
# 1 - Disallow cpu events for unpriv
# 2 - Disallow kernel profiling for unpriv

BENCHMARK_WORKLOAD="$@"
OUT_CMD="-o"
OUT_DIR="../../test_data/‘hostname‘/$OUT_FIL/‘date␣+%H-%M-%S-%m-%d
    -%Y‘"
mkdir -p $OUT_DIR

PERF_CNTRS="-e␣instructions␣-e␣cycles␣-e␣branches␣-e␣branch-misses"
COMMAND="$PERF␣$PERF_OPTS␣$PERF_CNTRS␣$RAPL_CNTRS␣$OUT_CMD␣$OUT_DIR
    /${OUT_FIL}_PERF␣$BENCHMARK_WORKLOAD"

TIME="date␣+%s,%N"
WATCH="watch␣-pn␣‘echo␣$INTERVAL␣/␣1000␣|␣bc␣-l‘"
if [ "$APM" -eq 1 ]; then
     eval $WATCH "\"$TIME >> $OUT_DIR/${OUT_FIL}_APM && $APM_CMD >>
        $OUT_DIR/${OUT_FIL}_APM\"␣&"
fi
if [ "$IPMI" -eq 1 ]; then
     eval $WATCH "\"$TIME >> $OUT_DIR/${OUT_FIL}_IPMI && $IPMI_CMD
        >> $OUT_DIR/${OUT_FIL}_IPMI\"␣&"
fi

echo $COMMAND >> $OUT_DIR/${OUT_FIL}
eval $COMMAND >> $OUT_DIR/${OUT_FIL}

# Kill the period IPMI and APM jobs
KILL="pkill␣-P␣$$"
eval $KILL

echo "Output␣written␣to␣$OUT_DIR/${OUT_FIL}"
```

Appendix C

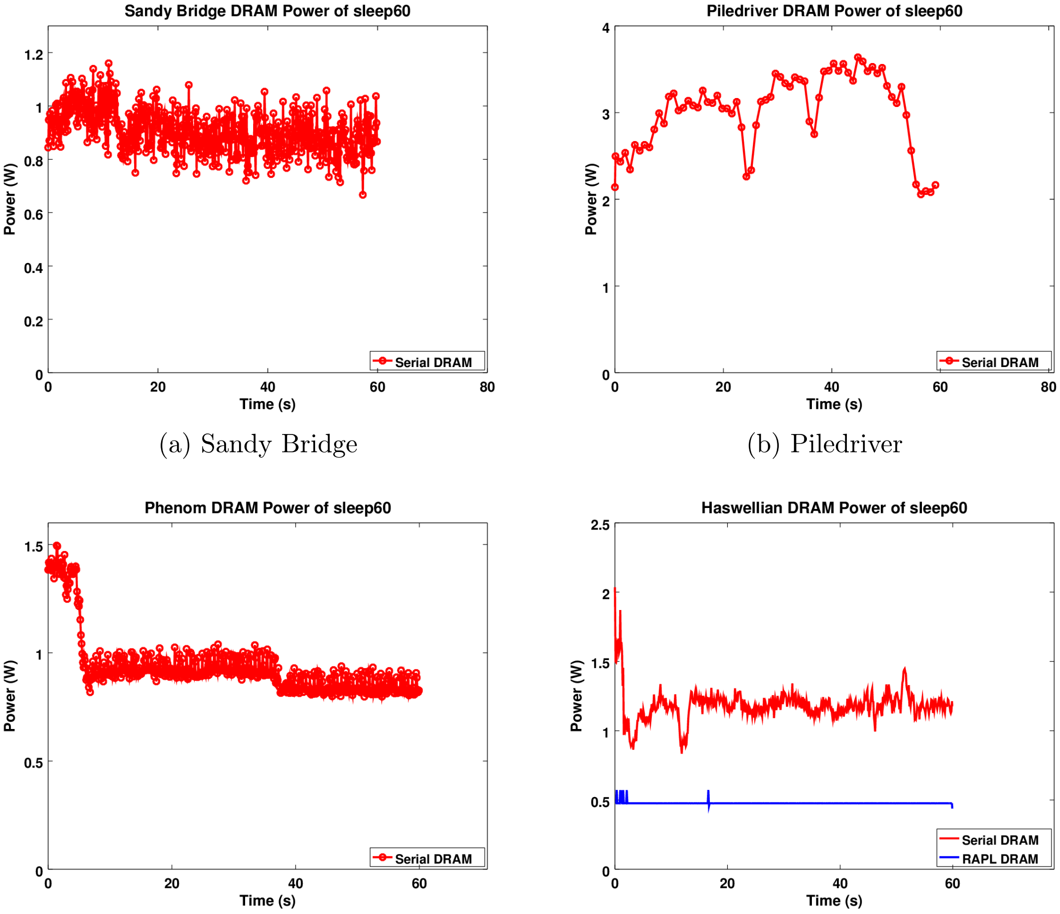
PARTS LIST

Table C.1: Parts List

Item	Quantity	Price ($)
Serial Data Logger		
Teensy 3.1	1	22.80
USB Micro Cable	1	5.99
Breadboard	1	5.95
Various Wires		
		Subtotal
		34.74
CS Instrumentation		
CS Extender Board	2	156.00
CS Tongue	1	22.25
32x2 Edge Connector	1	3.30
Ring Terminal Connectors	20	2.99
Quick Disconnect Pairs	20	2.99
16 AWG Stranged Wire	16 feet	3.20
ACS715 Hall Effect Sensor with Breakout Board	2	19.90
		Subtotal
		210.63
DRAM Instrumentation		
INA 122	1	6.64
Resistor	1	0.04
DIMM Extender	1	130.95
Breadboard	1	5.95
		Subtotal
		143.58
ATX Instrumentation		
P4 Extender	1	2.99
Quick Disconnect Pairs	20	2.99
		Subtotal
		5.98
		Total
		394.93

Appendix D

RESULTS



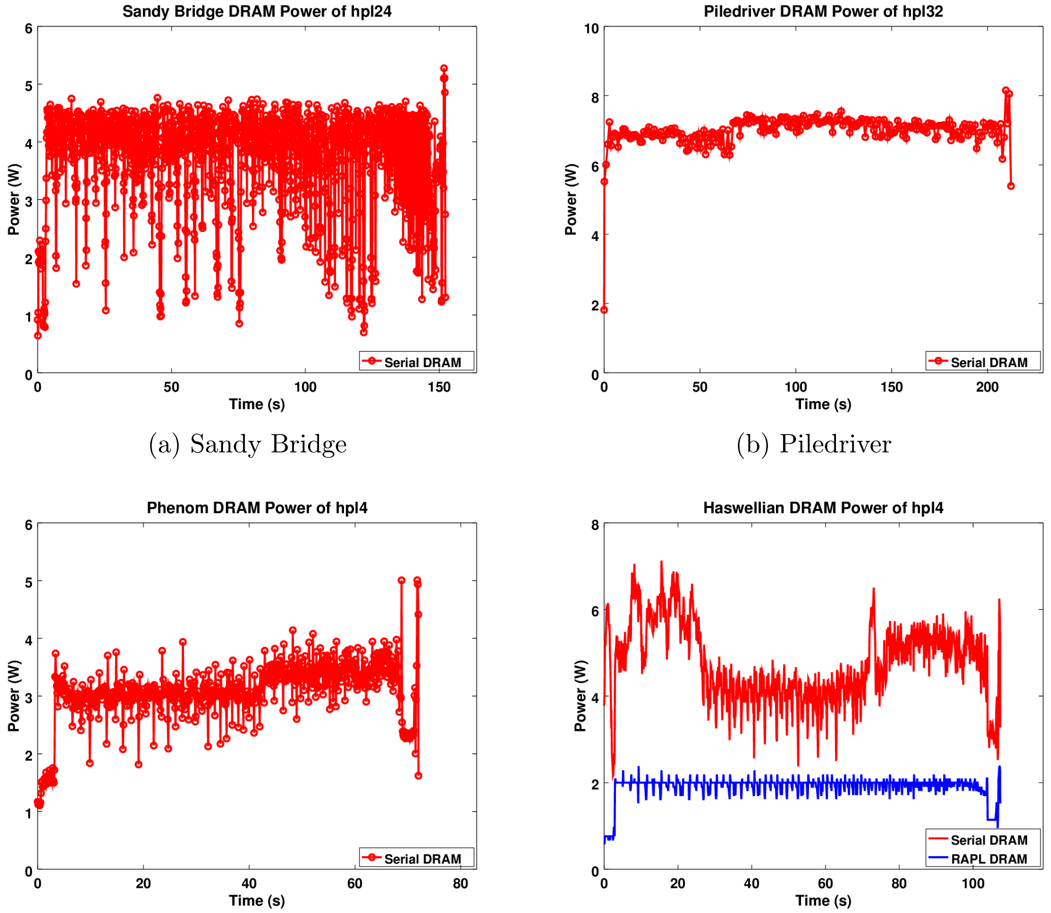
(a) Sandy Bridge

(b) Piledriver

(c) Phenom

(d) Haswellian

Figure D.1: Sleep DRAM on all Architectures



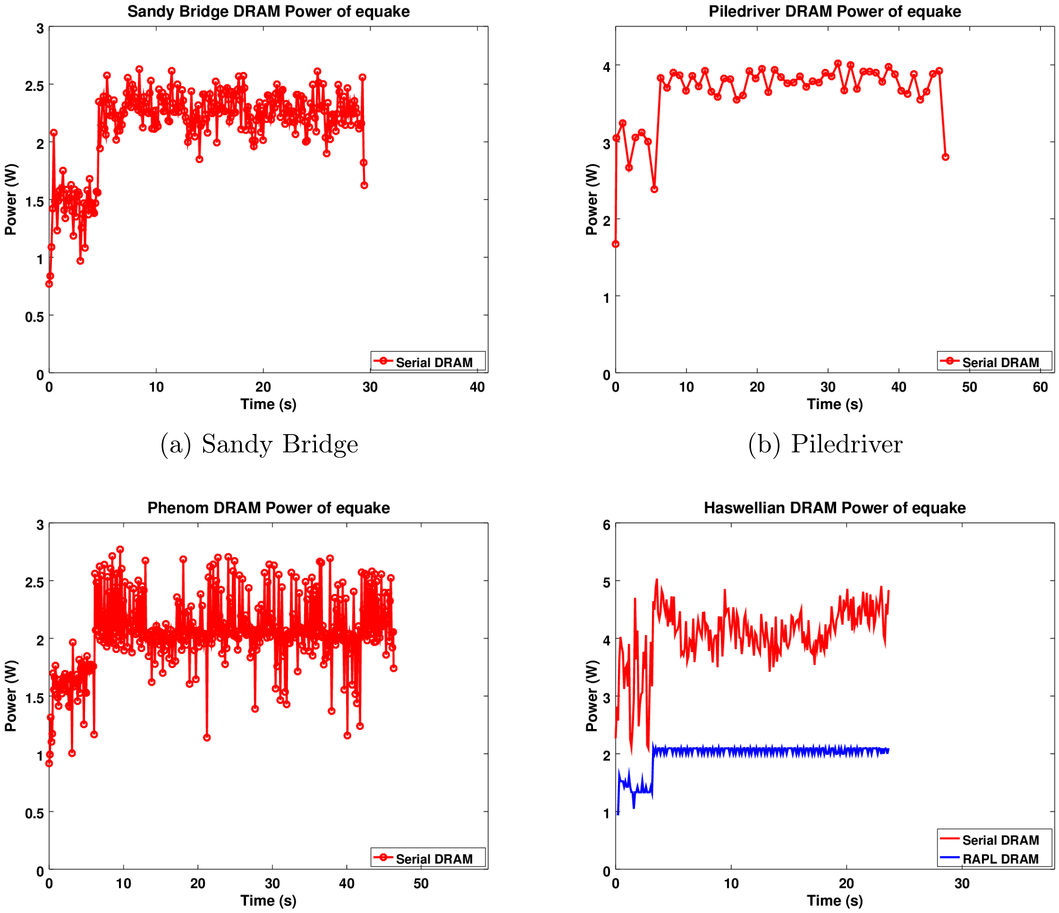
(a) Sandy Bridge

(b) Piledriver

(c) Phenom

(d) Haswellian

Figure D.2: High Performance Linpack DRAM all Architectures



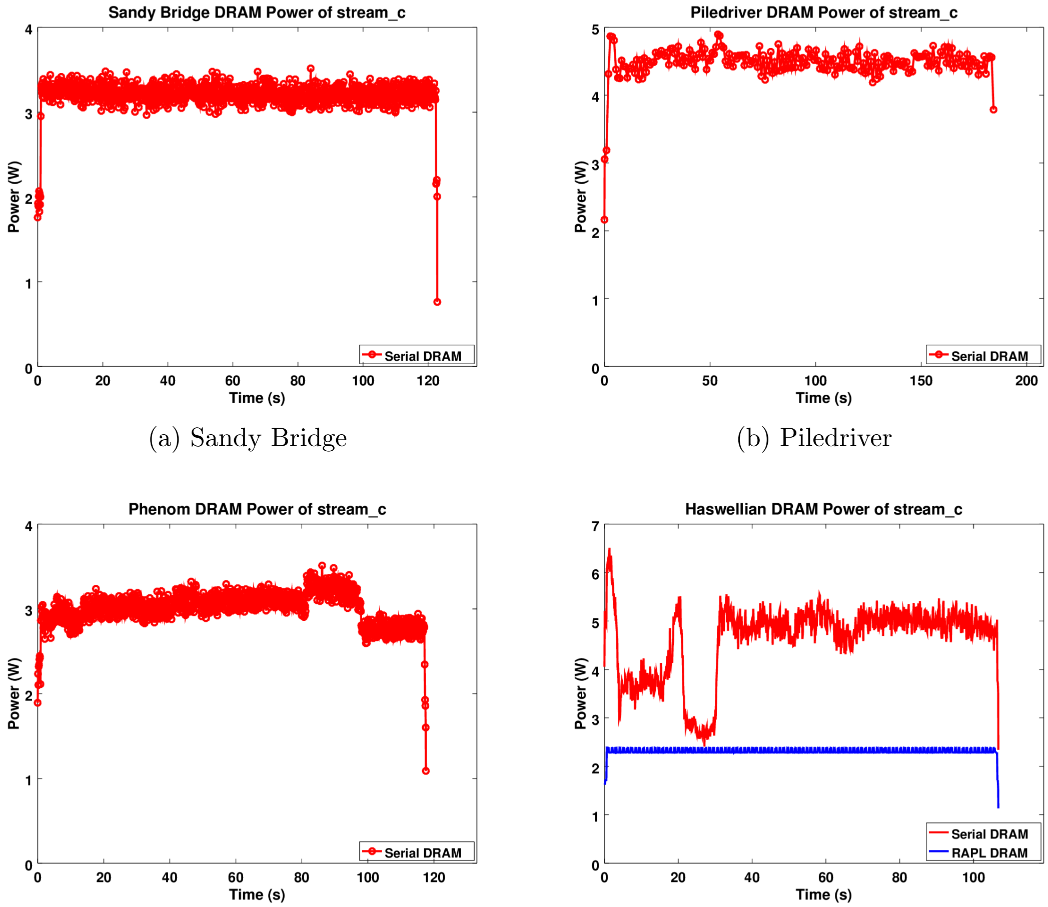
(a) Sandy Bridge

(b) Piledriver

(c) Phenom

(d) Haswellian

Figure D.3: Equake DRAM on all Architectures



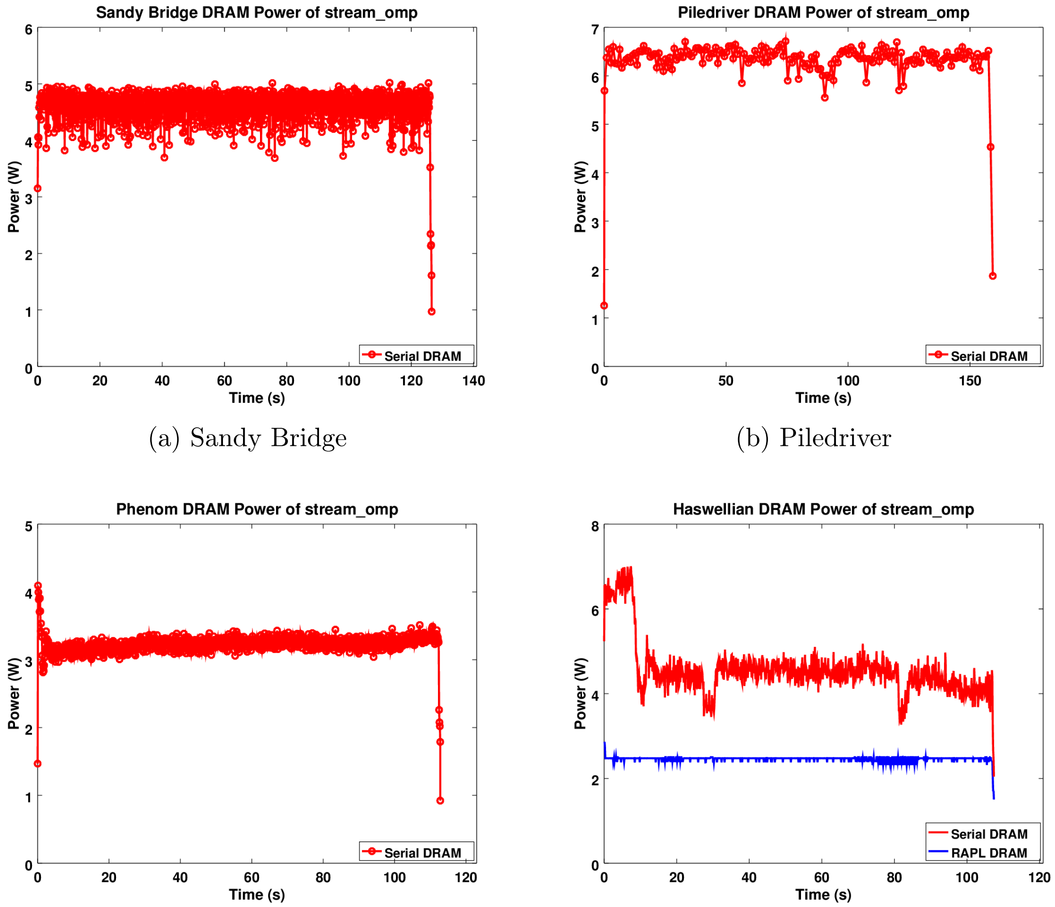
(a) Sandy Bridge

(b) Piledriver

(c) Phenom

(d) Haswellian

Figure D.4: Parallel STREAM DRAM on all Architectures



(a) Sandy Bridge

(b) Piledriver

(c) Phenom

(d) Haswellian

Figure D.5: STREAM on all Architectures

Appendix E

BENCHMARK INPUTS

```
HPLinpack benchmark input file
Innovative Computing Laboratory, University of Tennessee
HPL.out      output file name (if any)
6            device out (6=stdout,7=stderr,file)
1            # of problems sizes (N)
20480         Ns
1            # of NBs
256           NBs
0            PMAP process mapping (0=Row-,1=Column-major)
1            # of process grids (P x Q)
2            Ps
2            Qs
16.0         threshold
1            # of panel fact
2            PFACTs (0=left, 1=Crout, 2=Right)
1            # of recursive stopping criterium
4            NBMINs (>= 1)
1            # of panels in recursion
2            NDIVs
1            # of recursive panel fact.
1            RFACTs (0=left, 1=Crout, 2=Right)
1            # of broadcast
1            BCASTs (0=1rg,1=1rM,2=2rg,3=2rM,4=Lng,5=LnM)
1            # of lookahead depth
1            DEPTHs (>=0)
2            SWAP (0=bin-exch,1=long,2=mix)
64           swapping threshold
0            L1 in (0=transposed,1=no-transposed) form
0            U  in (0=transposed,1=no-transposed) form
1            Equilibration (0=no,1=yes)
8            memory alignment in double (> 0)
##### This line (no. 32) is ignored (it serves as a separator).
   ######
0                               Number of additional problem sizes
   for PTRANS
1200 10000 30000                values of N
0                               number of additional blocking sizes
    for PTRANS
40 9 8 13 13 20 16 32 64        values of NB
```

Figure E.1: Haswellian HPL.dat

```
HPLinpack benchmark input file
Innovative Computing Laboratory, University of Tennessee
HPL.out      output file name (if any)
6            device out (6=stdout,7=stderr,file)
1            # of problems sizes (N)
14000         Ns
1            # of NBs
128           NBs
0            PMAP process mapping (0=Row-,1=Column-major)
1            # of process grids (P x Q)
2            Ps
2            Qs
16.0         threshold
1            # of panel fact
2            PFACTs (0=left, 1=Crout, 2=Right)
1            # of recursive stopping criterium
4            NBMINs (>= 1)
1            # of panels in recursion
2            NDIVs
1            # of recursive panel fact.
1            RFACTs (0=left, 1=Crout, 2=Right)
1            # of broadcast
1            BCASTs (0=1rg,1=1rM,2=2rg,3=2rM,4=Lng,5=LnM)
1            # of lookahead depth
1            DEPTHs (>=0)
2            SWAP (0=bin-exch,1=long,2=mix)
64           swapping threshold
0            L1 in (0=transposed,1=no-transposed) form
0            U  in (0=transposed,1=no-transposed) form
1            Equilibration (0=no,1=yes)
8            memory alignment in double (> 0)
##### This line (no. 32) is ignored (it serves as a separator).
   ######
0                              Number of additional problem sizes
   for PTRANS
1200 10000 30000               values of N
0                              number of additional blocking sizes
    for PTRANS
40 9 8 13 13 20 16 32 64       values of NB
```

Figure E.2: Phenom HPL.dat

```
HPLinpack benchmark input file
Innovative Computing Laboratory, University of Tennessee
HPL.out      output file name (if any)
6            device out (6=stdout,7=stderr,file)
1            # of problems sizes (N)
24000         Ns
1            # of NBs
128           NBs
0            PMAP process mapping (0=Row-,1=Column-major)
1            # of process grids (P x Q)
2            Ps
16           Qs
16.0         threshold
1            # of panel fact
2            PFACTs (0=left, 1=Crout, 2=Right)
1            # of recursive stopping criterium
4            NBMINs (>= 1)
1            # of panels in recursion
2            NDIVs
1            # of recursive panel fact.
1            RFACTs (0=left, 1=Crout, 2=Right)
1            # of broadcast
1            BCASTs (0=1rg,1=1rM,2=2rg,3=2rM,4=Lng,5=LnM)
1            # of lookahead depth
1            DEPTHs (>=0)
2            SWAP (0=bin-exch,1=long,2=mix)
64           swapping threshold
0            L1 in (0=transposed,1=no-transposed) form
0            U  in (0=transposed,1=no-transposed) form
1            Equilibration (0=no,1=yes)
8            memory alignment in double (> 0)
```

Figure E.3: Piledriver HPL.dat

```
HPLinpack benchmark input file
Innovative Computing Laboratory, University of Tennessee
HPL.out      output file name (if any)
6            device out (6=stdout,7=stderr,file)
1            # of problems sizes (N)
25000         Ns
1            # of NBs
256           NBs
0            PMAP process mapping (0=Row-,1=Column-major)
1            # of process grids (P x Q)
1            Ps
24           Qs
16.0         threshold
1            # of panel fact
2            PFACTs (0=left, 1=Crout, 2=Right)
1            # of recursive stopping criterium
4            NBMINs (>= 1)
1            # of panels in recursion
2            NDIVs
1            # of recursive panel fact.
1            RFACTs (0=left, 1=Crout, 2=Right)
1            # of broadcast
1            BCASTs (0=1rg,1=1rM,2=2rg,3=2rM,4=Lng,5=LnM)
1            # of lookahead depth
1            DEPTHs (>=0)
2            SWAP (0=bin-exch,1=long,2=mix)
64           swapping threshold
0            L1 in (0=transposed,1=no-transposed) form
0            U  in (0=transposed,1=no-transposed) form
1            Equilibration (0=no,1=yes)
8            memory alignment in double (> 0)
##### This line (no. 32) is ignored (it serves as a separator).
   ######
0                               Number of additional problem sizes
   for PTRANS
1200 10000 30000                values of N
0                               number of additional blocking sizes
    for PTRANS
40 9 8 13 13 20 16 32 64        values of NB
```

Figure E.4: Sandy Bridge HPL.dat

Appendix F

EMBEDDED SOURCE CODE

```
#include "main.h"
#include "WProgram.h"

#include "ADC/ADC.h"
#include "usb_serial.h"
#include "my_defines.h"

#include <stdio.h>


ADC *adc;

extern "C" int main(void)
{
#ifdef USING_MAKEFILE
    // To use Teensy 3.0 without Arduino, simply put your code here
        .
    // For example:

    while (1) {
        adc_to_serial();
    }


#else
    // Arduino's main() function just calls setup() and loop()....
    setup();
    while (1) {
        loop();
        yield();
    }
#endif
}

ADC* adc_init(void) {
    ADC *adc = new ADC();
    adc->setAveraging(MY_ADC_AVG);
    adc->setResolution(MY_ADC_RES);
    adc->setConversionSpeed(ADC_HIGH_SPEED);
    adc->setSamplingSpeed(ADC_HIGH_SPEED);

    for (int i = 0; i < 10; i++) {
        pinMode(channel_list[i], INPUT);
    }


    return adc;
}

void serial_init(void) {
    Serial.begin(115200);
    Serial.setTimeout(-1);
}

void adc_to_serial(void) {
    char request = MY_SERIAL_NUL;
    int samples[sizeof(channel_list)];
    // Default to 1 second samples
    int interval = 1000;
    // Default to measuring only channel 1
    int channels = 1;
    // Network byte order
    bool nbo = false;

    serial_init();
    adc = adc_init();

    // Configure the LED
#define LED 13
    pinMode(LED, OUTPUT);

    // First receive a command byte
    // Process the command byte
    while (1) {
        request = Serial.read();

        switch (request) {
            case MY_SERIAL_INT:
                interval = get_interval();
                break;
            case MY_SERIAL_CHX:
                channels = get_channels();
                break;
            case MY_SERIAL_BEG:
                begin_sample(interval, channels, samples);
                break;
            case MY_SERIAL_REQ:
                transmit_sample(channels, samples);
                break;
            case MY_SERIAL_TBO:
                nbo = test_network_byte_order();
            case MY_SERIAL_END:
                break;
            case MY_SERIAL_NUL:
            default:
                break;
        }
    }

    delete adc;
}

// Time to delay between each round of samples
int get_interval(void) {
    return Serial.parseInt();
}

// Return a bitmap of the channels to use
int get_channels(void) {
    return Serial.parseInt();
}

void begin_sample(int interval, int channels, int *samples) {
    int channel = 0;

    // ADC library takes multiple samples automatically
    // Channel bitmap controls which channels to grab data from
    // /Max of twenty ADC channels
    for (unsigned int ch = 0; ch < sizeof(channel_list); ch++) {
        samples[ch] = 0;
    }

    for (int i = 0; i < MY_ADC_AVG; i++) {
        for (unsigned int ch = 0; ch < sizeof(channel_list); ch++)
           {
            if (channels&(1 << ch)) {
                channel = channel_list[ch];
                samples[ch] += ((int) adc->analogRead(channel))/
                   MY_ADC_AVG;
            }
        }
    delay(interval/(MY_ADC_AVG + 1)/2);
    digitalWriteFast(LED, HIGH);
    delay(interval/(MY_ADC_AVG + 1)/2);
    digitalWriteFast(LED, LOW);
    }
}

void transmit_sample(int channels, int *samples) {
    // int as uint*_t
    uint8_t *iasui;

    for (unsigned int i = 0; i < sizeof(channel_list); i++) {
        if (channels&(1 << i)) {
            // Write the int directly
            iasui = (uint8_t *) &(samples[i]);
            Serial.write(iasui, sizeof(int));
        }
    }
    Serial.printf("\r\n");
}

// teensy3.1 appears to be little endian by default
// Cortex M-4s can apparently switch between endianess though
// Client code writes byte 0 of an int to the server
// Compare received byte 0 and byte 0 of an int on server
bool test_network_byte_order(void) {
    int test_int = 1;
    char *test_char = (char *) &test_int;

    return (test_char[0] == Serial.read());
}

#ifndef MAIN_H_
#define MAIN_H_

#include "ADC/ADC.h"



void adc_to_serial(void);
void serial_init(void);
ADC* adc_init(void);
int get_interval(void);
int get_channels(void);
void begin_sample(int interval, int channels, int *samples);
void transmit_sample(int channels, int *samples);
bool test_network_byte_order(void);

#endif

#ifndef MY_DEFINES_H
#define MY_DEFINES_H

#include <stdint.h>



#define MY_SERIAL_NUL 'N'
#define MY_SERIAL_INT 'I'
```

```
#define MY_SERIAL_CHX ’C’
#define MY_SERIAL_BEG ’B’
#define MY_SERIAL_REQ ’R’
#define MY_SERIAL_TBO ’T’
#define MY_SERIAL_END ’E’

#define MY_ADC_AVG 8
#define MY_ADC_RES 16

#ifdef __arm__
#ifdef ADC_H
// List of ADC channels. Some of these are on the bottom of the board and hard to access
// Ten are easily accessible on the through hole pins
const uint8_t channel_list[] = {A0, A1, A2, A3, A4, A5, A6, A7, A8, A9, A10, A11, A12, A13, A14, A15, A16, A17, A18, A19, A20};
#endif
#else
const uint8_t channel_list[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20};
#endif

#endif /* end of include guard: MY_DEFINES_H */

75,79c75
<          // Make read() blocking
<          virtual int read() {
<                while (!this->available());
<                return usb_serial_getchar();
<          }
---
>          virtual int read() { return usb_serial_getchar(); }
```

BIOGRAPHY OF THE AUTHOR

Chad Michael Paradis is a candidate for the Master of Science degree in Computer Engineering from The University of Maine in August 2015.