

**A Measurement Platform
for DVS Algorithm Development and Analysis**

A Thesis
in TCC 402

Presented to

The Faculty of the
School of Engineering and Applied Science
University of Virginia

In Partial Fulfillment
of the Requirements for the Degree
Bachelor of Science in Computer Engineering

by
Arun Thomas
April 14, 2003

On my honor as a University student, on this assignment I have neither given nor received unauthorized aid as defined by the Honor Guidelines for Papers in TCC Courses.

Approved ----- (Technical Advisor)

Kevin Skadron

Approved ----- (TCC Advisor)

Rosanne Simeone

Contents

Glossary	iv
Abstract	1
1 Introduction	2
1.1 Problem Definition	2
1.2 Related Work	4
1.3 Project Contributions	5
1.4 Thesis Overview	5
2 Power Fundamentals	6
2.1 CPU Power and DVS	6
2.2 Power Measurement	7
3 Measurement System	10
3.1 Dynamic Voltage Scaling Platform	10
3.2 Measurement Platform	12
3.3 Web Serving Platform	14
4 Experiments and Results	16
4.1 Static Test	16
4.2 Dynamic Test	17
5 Conclusions	18
5.1 Interpretation	19
5.2 Recommendations	20
References	22
Bibliography	24
Appendix A: DVS algorithm	25

List of Figures

- 1.1 America Online’s datacenter [1]. 4
- 2.1 Voltage Measurement Method [2]. 8
- 2.2 Current Measurement Method [2]. 9
- 3.1 Measurement Platform Overview. 13
- 3.2 Circuit Overview (Adapted from [3]). 14

List of Tables

3.1	DVS system specifications	11
3.2	AMD Athlon XP 1800+ frequency and voltage pairs	11
4.1	Average Power at Various Performance Levels	17
4.2	Average Power With and Without DVS at Various Connection Rates	17

Glossary

Alternating Current (AC) – A term that is used to describe current that is bidirectional (i.e. alternates polarity). Electrical outlets provide AC current.

Ammeter – A tool used to measure the current flowing in a circuit.

Amplifier – A circuit that can be used to increase the power level of an electrical signal.

Clock Frequency – Determines the rate at which the CPU can complete computations

Clock Signal – The signal that synchronizes the operations that occur in a microprocessor.

CMOS – Complementary Metal-Oxide Semiconductor. The semiconductor technology used to create today's microchips.

Current (I) – The directed flow of current in a circuit, measured in Amperes (A).

Die Size – Overall physical area of a circuit

Direct Current (DC) – A term used to describe current that is unidirectional. Electrical devices accept DC current.

Dynamic Voltage Scaling (DVS) – A power saving technique that will dynamically lower a CPU's voltage based on the CPU's activity

HTTP – Hypertext Transfer Protocol, the language of the World-Wide Web.

Low-pass filter – A filter that passes all frequencies below a specified frequency with little or no loss, but strongly attenuates higher frequencies

Oscilloscope – A device used to measure the electrical characteristics of a circuit.

Power (P) – The amount of energy used per unit time, measured in watts (W)

Resistance – An opposition to current, measured in ohms.

Sense Resistor – A resistor that is used to indirectly determine the current in a circuit.

Voltage (V) – The potential that causes a directed flow of charge (current), measured in volts (V).

Voltmeter – A tool that is used to measure the voltage level of a circuit.

Web Server – A program that responds to HTTP requests (i.e. serves webpages)

Abstract

Excessive power consumption in a microprocessor has several negative impacts: a decrease in system reliability, an increase in both the initial cost and the operation expenses of computers, and a reduction in the battery life of power-critical portable and handheld systems. Dynamic voltage scaling is a technique used to solve these and other power-related CPU problem by lowering the CPU voltage dynamically, when doing so will not significantly impact system performance. The CPU voltage is quadratically related to CPU power, so this can lead to a significant power savings. The system described in this report is designed to be a power measurement platform for DVS algorithms that optimize the power efficiency of webserver workloads. IBM's Austin research lab [4] first made the claim that DVS could be a useful technique to manage power in server systems, since servers experience long periods of inactivity. They were able to prove their claim using simulated data. This measurement system can be used to validate their work and build upon it using real-world experimental data. This research project produced a usable power measurement system. This measurement platform will allow researchers at UVA to more effectively develop DVS algorithms. Moreover, this has produced real-world power data in a modern server-class DVS system. Such data had not existed before in the research literature. Ultimately, this work may lead to better DVS algorithms and thus more power efficient servers, improving the overall quality of electronic services that depend on these servers.

Chapter 1

Introduction

Excessive power consumption in a microprocessor has several negative impacts: a decrease in system reliability, an increase in both the initial cost and the operation expenses of computers, and a reduction in the battery life of power-critical portable and handheld systems. There are a number of techniques to reduce power consumption, but this project focuses on dynamic voltage scaling (DVS). Dynamic voltage scaling is a technique to reduce the power consumption of a microprocessor by lowering the CPU voltage dynamically, when doing so will not significantly impact system performance. In previous efforts researchers have constructed simulations to model DVS behavior. This project has broken from this tradition and produced a measurement platform that captures real-world power data from a modern DVS-capable system, specifically an AMD mobile Athlon machine. This captured power data was subsequently used to develop and verify a simple DVS algorithm.

1.1 Problem Definition

Traditionally, microprocessor designers focused all of their efforts on optimizing two CPU characteristics: CPU performance and die size (i.e. overall chip area). There are sound reasons for the original bias toward these parameters. As software engineers work

to bring more useful and complex applications to the public, the performance capability of CPUs must increase in order to facilitate the increased complexity. Die size, or overall chip area, is also an important constraint, since larger dies lead to higher manufacturing costs and ultimately higher prices for the consumer. Recently, power has become the third chief design characteristic for microprocessors, so there is much interest in power management techniques such as DVS [5] [6] [7].

There are several reasons for this new interest in DVS. First, portable and handheld systems (e.g. PDAs, cellular phones) have limited battery usage, so a power savings will increase battery life. Second, a reduction in power consumption leads to increased reliability in microprocessors. Higher power devices dissipate more heat. Excessive heat can permanently damage a chip, so lowering a processor's power reduces the likelihood of failure [8]. Third, reducing power can reduce cost. Processor designers build their cooling systems for peak thermal characteristics that are far more stringent than the average thermal behavior, though the peak value is almost never reached. Power management techniques can be used to indirectly regulate the thermal characteristics such that the peak thermal value is much closer to the average value, thus allowing designers to employ less costly cooling systems at the CPU level [9].

The problems of excessive power consumption are magnified for datacenters with hundreds or thousands of server machines. A typical datacenter will house hundreds or thousands of machines, densely packed in order to maximize floor space. America Online's server farm, a representative example, is depicted in Figure 1.1. This form of server consolidation can lead to increased reliability concerns unless more expensive cooling systems are employed [4]. Furthermore, power and cooling costs comprise a significant part of a datacenter's budget. For example, a datacenter with 8,000 servers can consume 2,000,000 watts of power, which accounts for 25% of the operating cost of such a facility [10]. Furthermore, the server racks, in many cases, cannot be fully populated, since the rack cannot power all its constituent servers. Thus, datacenters must purchase a larger physical area to

house the same number of servers, increasing costs. These problems will only worsen if the trend of newer server machines offering higher performance levels at the cost of increased power consumption continues [4].



Figure 1.1: America Online’s datacenter [1].

1.2 Related Work

There is a significant body of research on applying DVS to compilation, real-time scheduling, and embedded systems applications [11] [12] [8] [13] [14]. IBM’s Austin Research Lab [4] was the first to suggest that DVS should not be restricted to handheld devices as it had been previously, but should be extended to server machines. The IBM group determined experimentally that the CPU is the highest power consumer in server systems. They also demonstrated that web servers experience long periods of low utilization and, thus, are prime candidates for voltage scaling. Consequently, they built a simulator application that

models a DVS-capable system for webserver workloads. Using real server workloads as input to the simulator, they were able to show that DVS can reduce power consumption by up to 36%.

1.3 Project Contributions

This project has several contributions. First, it produced a measurement platform that can be used by researchers at UVA in order to design and analyze the effectiveness of their DVS techniques. Although the power measurement system is geared towards server applications, it can be readily adapted to other applications broadening the impact of this research. Second, it has produced real-world power data that clearly shows that DVS can be an effective technique for reducing power consumption. Third, this research effort produced a rudimentary DVS algorithm used to validate the measurement platform. This algorithm can be used as a basis for further DVS algorithm research.

1.4 Thesis Overview

This report begins by covering the fundamentals of CPU power consumption and power measurement in Chapter 2. It then covers the experimental setup that is used to obtain power measurements of the DVS system in Chapter 3. Chapter 4 describes the experiments used to test the functionality of the measurement system, as well as the produced results. The report concludes in chapter 5.

Chapter 2

Power Fundamentals

This chapter will provide a brief introduction to power measurement fundamentals. It will define some fundamental terms and prepare the reader for the discussion of the measurement platform in the following chapter. Furthermore, this chapter will further explore the relationship between DVS and power consumption.

2.1 CPU Power and DVS

In order to fully understand DVS's effect on power consumption, it is important to look at the power equation (2.1) for all CMOS (Complementary Metal-Oxide Semiconductor) circuits, including microprocessors. Power in CMOS circuits has two components: a dynamic component which is caused by 0-to-1 and 1-to-0 logic transitions and a static component that exists regardless of whether the circuit switches logic states. Dynamic Voltage Scaling only attacks $P_{dynamic}$; there are a number of other power saving techniques that address P_{static} . Equation 2.2 shows the factors that make up $P_{dynamic}$. The capacitive load (C_L) is an intrinsic property of the circuit. The switching probability (c) determines how often the circuit will make a transition. Neither of these factors are related to DVS [15] [16].

$$P_{avg} = P_{dynamic} + P_{static} \quad (2.1)$$

$$P_{dynamic} = cC_L f V_{dd}^2 \quad (2.2)$$

Dynamic Voltage Scaling is able to effectively reduce power consumption because of the other two factors in Equation 2.2. The CPU's voltage (V_{dd}) is quadratically related to power - a reduction in the CPU's voltage supply leads to an inverse squared reduction in power. There is one important caveat: reducing the CPU voltage requires a corresponding decrease in the clock frequency (f) of the processor for correct operation. This has a positive impact on power consumption, since scaling frequency and voltage down concurrently leads to an inverse cubed reduction in power. On the other hand, lowering the CPU's clock frequency leads to decreased performance. The clock is a signal that synchronizes the operations that occur in a microprocessor. Reducing the clock frequency will slow down the processor. Thus, a decrease in the clock frequency of the CPU will cause a corresponding decrease in processor performance. Fortunately, the CPU rarely needs to operate at peak efficiency. Consequently, the CPU may operate at a reduced voltage level at these times, saving power without negatively impacting performance. [6].

2.2 Power Measurement

Power (P), the amount of energy a system uses per unit time, cannot be measured directly. Instead, power measurements must be derived from two other electrical characteristics: current and voltage. Current (I) is defined as the directed flow of current in a circuit. Voltage (V) is defined as the potential that causes a directed flow of charge (current). Current and voltage are related to power via Equation 2.3.

$$P = IV \quad (2.3)$$

Voltage is measured directly by connecting a voltmeter across the load, as shown in Figure 2.1. One of the voltmeter's leads is connected to one side of the component, while the other lead is connected to the other side of the component.

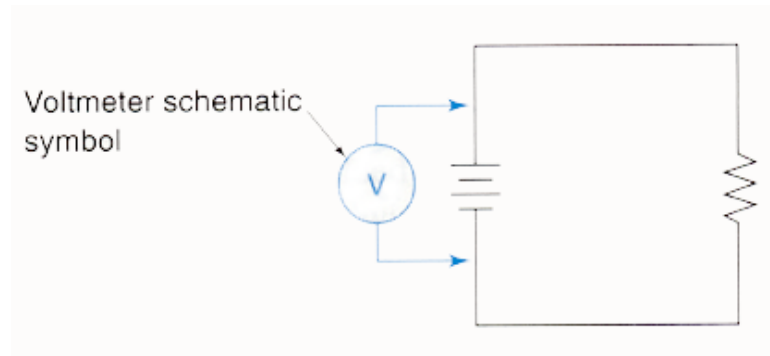


Figure 2.1: Voltage Measurement Method [2].

There are two methods to measure the current in a circuit. First, current can be measured by inserting an ammeter in the current path. Thus, the circuit must be broken and the meter must be inserted so that current flows through the meter as shown in Figure 2.2. Second, current may be measured indirectly by breaking the circuit, connected a sense resistor in series with the circuit, and measuring the voltage drop across the sense resistor. A sense resistor is a resistor with a small, know resistance (R) value. Since the sense resistor's resistance value is known, the current of the circuit can subsequently be determined using Equation 2.4. Actually, this is what the ammeter does internally [2].

$$I = V/R \quad (2.4)$$

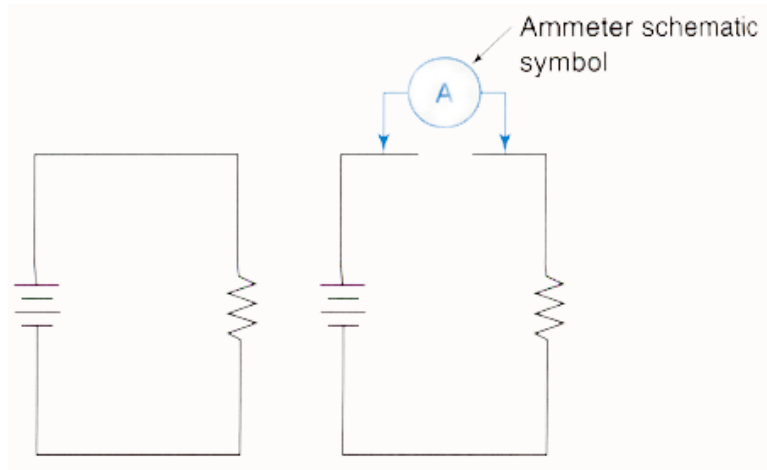


Figure 2.2: Current Measurement Method [2].

It is important to use a small sense resistor value. A larger resistor value will decrease the accuracy of power measurements, since power will be dissipation across the resistor. Equation 2.5 shows that an increase in the resistance value (R) while holding the voltage across the resistor constant will lead to an increase in the power consumed across the resistor and thus will lead to error in our measurements [2].

$$P = V^2 R \quad (2.5)$$

Now that the fundamentals of power measurement have been reviewed, the measurement system's design can be discussed.

Chapter 3

Measurement System

This chapter will provide an overview of the measurement system developed in this project. Section 3.1 will cover the hardware and software platform used for DVS. Section 3.2 will explain the measurement methodology. Section 3.3 will discuss the web-serving platform as well as the platform used to simulate web clients.

3.1 Dynamic Voltage Scaling Platform

Determining a suitable platform for DVS study required considerable research. There are a number of possible CPU options for DVS systems: Transmeta's Crusoe, Intel's mobile Pentium, and AMD's mobile Athlon. Crusoe-based systems were ruled out, since they cannot meet the high-performance requirements of a server-class system. Systems based on the mobile Pentium were also ruled out, since Intel's Speedstep technology only allows for two performance levels. This would hamper the effectiveness of DVS algorithms, since the limited number of performance levels translates to fewer opportunities for power saving by lowering CPU voltage and/or frequency. The mobile Athlon XP was finally chosen, since it would allow for the most flexibility in terms of performance levels. Moreover, a number of manufacturers create high performance systems based on the mobile Athlon processor.

The Compaq Evo n1015v, an Athlon XP based system, served as the DVS system whose power would be measured. An actual server system would have been preferable, since this research project is geared towards server-class systems. At present there are no desktop or server systems that incorporate DVS, so a laptop had to serve the purpose. Nevertheless, the Evo should prove more than adequate, since it performs very well. This is clear from the system specifications shown in Table 3.1. This system supports five performance levels (i.e. frequency/voltage pairs) as seen in Table 3.2.

Processor	AMD Mobile Athlon XP 1800+ (1533/266 MHz)
Northbridge Chipsets	IGP 320 M north bridge
Southbridge Chipsets	ALi M1535+
RAM	256MB PC2100 SDRAM
Hard Disk	Toshiba MK4018GAP ATA-5 40GB/4200rpm/2MB
CD/DVD ROM	Toshiba SDR2102 (ATA-2) 8x/8x8x24xDVD/CDRW Combo
Graphics Chipset	Integrated, 64 MB video memory
Display Size	15.1"
Audio	SoundMax Integrated Digital Audio
Ethernet	Realtek 8139C 10/100 MBit Ethernet card
AC Adapter	18.5V/4.9A DC

Table 3.1: DVS system specifications

Performance Level	Speed (MHz)	Voltage (V)
1	533	1.20
2	666	1.20
3	800	1.20
4	1200	1.35
5	1533	1.50

Table 3.2: AMD Athlon XP 1800+ frequency and voltage pairs

The DVS system runs the testing version of Debian GNU/Linux operating system. GNU/Linux is an ideal research platform, since a significant number of the servers on the Internet run GNU/Linux. Moreover, GNU/Linux is the only operating system with drivers that allow for the manual adjustment of CPU performance level. Without this capability any research endeavor into DVS would prove impossible.

The DVS system is running a development version of the Linux kernel (2.5.65), since it has the most recent frequency/voltage scaling drivers. Using the Linux scaling drivers, CPU performance levels can be set by writing to relevant files in the `sysfs` filesystem, a filesystem that exposes Linux kernel data structures to user applications. The 2.5 development series is technically considered unstable, yet the use of this kernel has not hindered this research effort at all.

3.2 Measurement Platform

Accurate measurement of power is an important part of this research. Consequently, significant time was devoted to devising and constructing the power measurement platform. The measurement platform consists of three main parts: a measurement PC, a digital oscilloscope, and a breadboard with custom circuitry. An overview of the entire measurement platform is shown in Figure 3.1. The DVS laptop normally receives power through its AC adapter, which is connected to an electrical outlet. The AC adapter converts the alternating polarity current (AC) that the wall socket produces and converts it to a direct (i.e. fixed polarity) current that electrical components can use. Chapter 2 explained that current measurements require breaking the circuit path in order to place a sense resistor in series with the circuit to be measured. That is precisely what has been done here. The AC adapter has been cut and the ends have been spliced into the custom circuit board. The AC adapter must be cut on the DC side of the AC adapter in order to measure the current of the laptop. The digital oscilloscope digitizes the current and voltage data and sends this data to the Measurement PC. The measurement PC is running custom software that computes power data using the signals it receives from the digital oscilloscope. Furthermore, the measurement PC will log the power data to a file. The measurement system/digital oscilloscope combination is able to take 80 power readings per second.

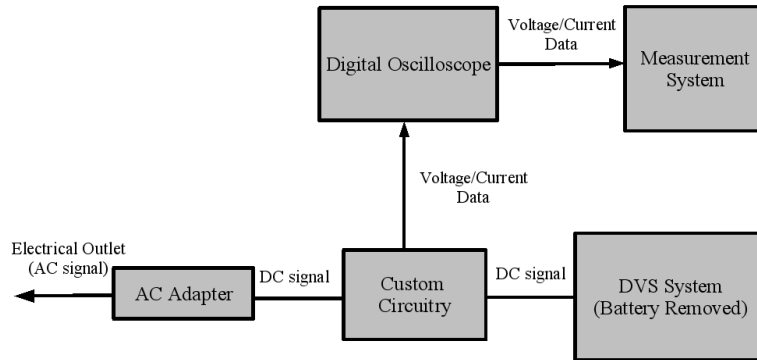


Figure 3.1: Measurement Platform Overview.

Now that the measurement platform has been discussed as a whole, the power measurement circuitry will be explained. Figure 3.2 shows the main components of the circuit board. A sense resistor with a value of 0.01 ohms is used. The voltage drop measured across the sense resistor is amplified by the LT1787 current sense amplifier and sent to the pin labeled output. This voltage can be measured by placing probes at both the output node and the ground node. This voltage can then be used to determine the current via Equation 2.4. This equation is determined by taking into account the sense resistor's resistance value and the amplifier gain (i.e. level of amplification). The laptop's system voltage level can be measured by placing one probe at the node that connects to the laptop and another probe at ground. Power can be determined using voltage and current using Equation 2.3. The voltage regulator allows the circuit board to derive power from the AC adapter. This will introduce slight error into the power measurements, since the power consumption of the circuit board is also measured. This error is negligible, since the laptop system consumes considerably more power than the circuit board. A capacitor (i.e. low pass filter) is connected across the output pin in order to filter out noise.

$$I = 0.8 * Output \quad (3.1)$$

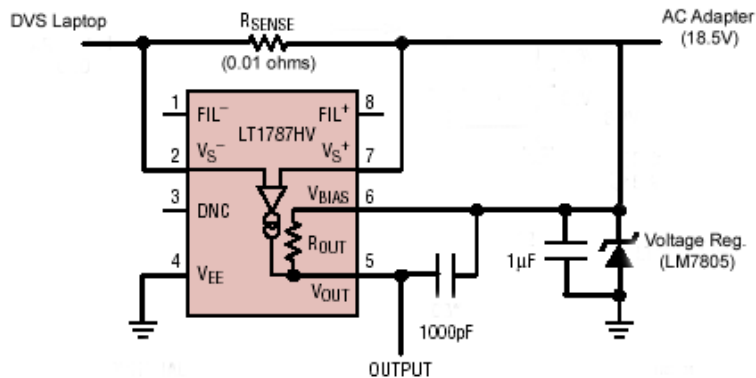


Figure 3.2: Circuit Overview (Adapted from [3]).

It is important to note that this system measures the power of the overall laptop system, not just the CPU. Measuring CPU power would require opening up the laptop case and modifying the motherboard. Unfortunately, there was not enough time for this, nor was there a spare laptop free for experimentation. Consequently, some of the interesting DVS effects on power will be masked by the power characteristics of other components of the system. Nevertheless, useful data was still collected.

3.3 Web Serving Platform

Since this measurement platform is geared towards webserver applications, it is necessary to run a webserver application on the DVS laptop. For this purpose the Apache webserver was chosen, since it is the most popular webserver on the Internet and it runs well on GNU/Linux systems. In the future the measurement system will be adapted to use Apache 2.0.

A tool called Httperf (version 0.8) was used to simulate the effect of numerous clients connecting to the Apache webserver running on the DVS system. Httperf provides a flexible facility for generating various HTTP workloads and for measuring server performance. The primary use of this tool was to generate workloads for the webserver, thus varying the activity of the CPU and subsequently providing opportunities for DVS tech-

niques to save power. Httpperf makes it possible to simulate hundreds of web clients using only one computer.

Chapter 4

Experiments and Results

This chapter will discuss the experiments conducted using the DVS measurement platform and the real-world power data that it has produced.

4.1 Static Test

After the DVS system was fully implemented, the first real test involved varying the CPU's performance level manually. The average power consumption was measured both when the CPU was mostly idle and when the CPU had a utilization of 100%. The data in Table 4.1 clearly shows that the measurement system is able to observe variances in system power due to changes in the performance level. It is interesting to note that the power readings do not change drastically when an idle CPU's performance level is changed. On the other hand, a fully utilized CPU's power level changes dramatically as the performance level is changed. The changes are most noticeable between voltage steps (e.g. the shift from performance level 3 to performance level 4). This is to be expected due to the quadratic relation between power and voltage level.

Speed (MHz)	Voltage (V)	Idle CPU Avg. Power (W)	100% CPU Util. Avg. Power (W)
533	1.20	24.45	34.20
666	1.20	24.64	36.62
800	1.20	24.65	39.22
1200	1.35	26.71	54.47
1533	1.50	29.71	80.85

Table 4.1: Average Power at Various Performance Levels

4.2 Dynamic Test

A more thorough test of the measurement platform was conducted when an actual DVS algorithm executed on the laptop while running a webserver. The DVS algorithm used in this test was adapted from a program written by Carl Thompson. The algorithm is fairly simplistic; it raises or lowers the CPU performance level based on what percentage of the time the CPU is idle. If the CPU is idle for more 75% of the time, the performance level decreases to the next level. If the CPU is idle for less than 25% of the time, the performance level increases to the next level. These thresholds were arbitrarily chosen. This algorithm is merely used for validation purposes, though it could be used as the foundation for future DVS work.

Table 4.2 shows the result of this experiment. The power saving that DVS provides diminishes as the connection rate increases. This is to be expected, since a CPU's activity rate increases with the connection rate. Thus, there a fewer opportunities to scale down voltage.

Connection Rate (conn./sec)	Avg. Power (W)	Avg. Power w/DVS (W)
100	38.48	26.49
200	42.18	29.69
300	55.91	53.45
400	72.43	70.75
500	74.26	73.37

Table 4.2: Average Power With and Without DVS at Various Connection Rates

Chapter 5

Conclusions

This project has produced a power measurement platform and methodology for DVS-enabled webserver systems. The measurement platform is comprised of a digital oscilloscope, custom circuitry, and a PC that processes and logs power data to disk. The system can take 80 power readings per second.

A modern DVS-enabled laptop machine serves as the test subject of this measurement system. The DVS-enabled laptop runs the Apache webserver software. Httperf generates simulated web clients that connect to Apache webserver, providing stimulus for the DVS power efficiency tests. In this webserver power measurement methodology, power data is collected as the connections rate (i.e. number of web clients connecting to the server per second) increases.

In addition, this project has produced real-world power data for a modern DVS-capable system. The system ran both static and dynamic scaling tests. In each trial of the static test, the CPU was manually placed into one of the five performance levels and remained at that level for the duration of the test. The system power was measured when the CPU was in both an idle and a fully loaded state for each of the performance levels. As expected the data from this test (found in Table 4.1) shows that lowering the CPU's voltage level can lower the system's power level dramatically. The second test is more relevant to

the project objective, since the test produced power consumption data for a system acting in a webserver capacity. The data from this test (Table 4.2) shows DVS is able to save power in webserver systems at lower connection rates.

Furthermore, a basic DVS algorithm has been developed in order to verify the measurement platform. This DVS laptop runs this algorithm in the dynamic webserver test. This algorithm will scale the voltage based on the what percentage of the time the CPU is idle.

5.1 Interpretation

This project has met its goal of creating a measurement system that obtains real-world power data of a DVS-capable system under a webserving workload. This is significant since all previous power data for DVS-capable server systems has relied on simulation. This is the first project to produce real-world power data for such systems. Consequently, this system can be used to validate the Austin Research Lab's claim that DVS is an effective technique for power reduction in webserver systems. Indeed, the results shown in Table 4.2 suggest that IBM's claim is true. Further testing needs to be done, however, to determine how effectively DVS can save power under real web traffic workloads. Currently, Httperf is used to generate simulated client traffic.

There are some other limitations to the current power measurement methodology. The system only measures overall system power. It would be helpful to measure CPU power as well. DVS works to lower the CPU's power consumption. Thus, measuring CPU power separately would show the effects of DVS more clearly, since system power aggregates the power characteristics of all the components in the system. Consequently, events unrelated to DVS (e.g. disk drives powering up) can distort the power consumption picture given by system power.

The lack of CPU power measurements should not significantly impact the conclu-

sions of this paper, since the tests that were run are CPU-intensive. Thus, the majority of the power difference can be attributed to CPU power consumption and thus DVS' effects. Furthermore, the CPU is the highest power consumer in the system, so it will affect the power characteristics of the system more than other components. In fact, the results show that DVS does reduce CPU power. Moreover, the captured power data clearly shows that DVS can affect system power dramatically.

5.2 Recommendations

This research presents several opportunities for future work. First, the power measurement methodology can be improved by instrumenting the system to measure CPU power. In addition, the oscilloscope can be replaced with a data acquisition card that would allow for a higher sampling rate (i.e. faster collection of data), also improving the accuracy of results. The dynamic test could employ more realistic traffic loads by using web server logs from actual websites. In order to truly measure the effectiveness of a DVS-enabled webserver, the server system should respond to more realistic web traffic. Httpperf can be configured such that it reads traffic logs from actual websites. Httpperf simulates webserver traffic fairly well, but real traffic is still preferred.

Additionally, this project focused primarily on measuring the power efficiency of DVS-enabled webserver systems, though it should be extended to measure the performance of these systems as well. Generally, performance is a more important design constraint than power consumption in webserver systems. Few datacenters will deploy DVS-enabled systems if performance is impacted significantly. Httpperf can be readily used for this purpose. Unfortunately, there was not enough time to incorporate this into the project.

The measurement system could also be extended to provide power data for other applications such as databases and graphics-intensive programs. Furthermore, the rudimentary DVS algorithm used for system validation could be developed into full-fledged

DVS algorithm. The performance measurement additions to the measurement platform would ultimately be necessary for DVS algorithm development, since an effective DVS algorithm strikes the right balance between performance and power savings.

Though there are still many improvements that can be made to the system, researchers at UVA can use this system in their DVS algorithm research. Computer scientists at other institutions can use the methodology described in this report in order to develop similar systems of their own.

The measurement platform described in this project will facilitate the development of more effective DVS algorithms. Eventually, these new DVS algorithms may find their way into the server systems of large electronic services companies such as America Online. Ultimately, these new DVS-enabled servers would allow such companies to reduce their operating costs and increase the reliability of their services thereby reducing consumer prices and improving the end user's online experience.

References

- [1] Steve Lohr and John Markoff, “Digital engine rooms: Computing centers become the keeper of web’s future,” *New York Times*, vol. A2, May 19, 1999.
- [2] Robert Paynter, *Introductory electric circuits*, Prentice Hall, Upper Saddle River, NJ, 1999.
- [3] Linear Technology Corporation, *LT1787/LT1787HV datasheet*, 1999.
- [4] Pat Bohrer, Elmootazbellah Elnozahy, Tom Keller, Michael Kistler, Charles Lefurgy, and Ram Rajamony, “The case for power management in web servers,” *Power Aware Computing*, 2002.
- [5] Trevor Mudge, “Power: A first-class architectural design constraint,” *IEEE Computer*, vol. 34, no. 4, April 2001.
- [6] Anantha Chandrakasan and Robert Brodersen, “Minimizing power consumption in digital cmos circuits,” in *IEEE*, April 1995, vol. 83, pp. 498–523.
- [7] Vivek Tiwari, Deo Singh, Suresh Rajgopal, Gaurav Mehta, Rakesh Patel, and Franklin Baez, “Reducing power in high-performance microprocessors,” in *Design Automation Conference*, 1998, pp. 732–737.
- [8] Chung-Hsing Hsu and Ulrich Kremer, “Compiler-directed dynamic voltage scaling for memory-bound applications,” .
- [9] Kevin Skadron, Tarek Abdelzaher, and Mircea Stan, “Control-theoretic techniques and thermal-RC modeling for accurate and localized dynamic thermal management,” Tech. Rep. CS-2001-27, University of Virginia, 2001.
- [10] Deo Singh and Vivek Tiwari, “Power challenges in the internet world,” in *Cool Chips Tutorial, held in conjunction with 32nd Annual International Symposium on Microarchitecture*, November 1999.
- [11] Padmanabhan Pillai and Kang G. Shin, “Real-time dynamic voltage scaling for low-power embedded operating systems,” in *ACM Symposium on Operating Systems Principles*, 2001, pp. 89–102.
- [12] Erven Rohou and Michael Smith, “Dynamically managing processor temperature and power,” in *2nd Workshop on FeedbackDirected Optimization*, November 1999.

- [13] Jason Flinn and M. Satyanarayanan, “Energy-aware adaptation for mobile applications,” in *Symposium on Operating Systems Principles*, 1999, pp. 48–63.
- [14] Tajana Simunic, Luca Benini, Peter W. Glynn, and Giovanni De Micheli, “Dynamic power management for portable systems,” in *Mobile Computing and Networking*, 2000, pp. 11–19.
- [15] Jan Rabaey, *Digital Integrated Circuits: A Design Perspective*, Prentice-Hall, 1996.
- [16] David Patterson and John Hennessy, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann, San Mateo, 1990.

Bibliography

Pat Bohrer, Elmootazbellah Elnozahy, Tom Keller, Michael Kistler, Charles Lefurgy, and Ram Rajamony, “The case for power management in web servers,” *Power Aware Computing*, 2002.

Anantha Chandrakasan and Robert Brodersen, “Minimizing power consumption in digital CMOS circuits,” in *IEEE*, April 1995, vol. 83, pp. 498–523.

Jason Flinn and M. Satyanarayanan, “Energy-aware adaptation for mobile applications,” in *Symposium on Operating Systems Principles*, 1999, pp. 48–63.

Chung-Hsing Hsu and Ulrich Kremer, “Compiler-directed dynamic voltage scaling for memory-bound applications,” .

Linear Technology Corporation, *LT1787/LT1787HV datasheet*, 1999.

Steve Lohr and John Markoff, “Digital engine rooms: Computing centers become the keeper of web’s future,” *New York Times*, vol. A2, May 19, 1999.

Trevor Mudge, “Power: A first-class architectural design constraint,” *IEEE Computer*, vol. 34, no. 4, April 2001.

David Patterson and John Hennessy, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann, San Mateo, 1990.

Robert Paynter, *Introductory electric circuits*, Prentice Hall, Upper Saddle River, NJ, 1999.

Padmanabhan Pillai and Kang G. Shin, “Real-time dynamic voltage scaling for low-power embedded operating systems,” in *ACM Symposium on Operating Systems Principles*, 2001, pp. 89–102.

Jan Rabaey, *Digital Integrated Circuits: A Design Perspective*, Prentice-Hall, 1996.

Erven Rohou and Michael Smith, “Dynamically managing processor temperature and power,” in *2nd Workshop on Feedback-Directed Optimization*, November 1999.

Tajana Simunic, Luca Benini, Peter W. Glynn, and Giovanni De Micheli, “Dynamic power management for portable systems,” in *Mobile Computing and Networking*, 2000, pp. 11–19.

Deo Singh and Vivek Tiwari, “Power challenges in the Internet world,” in *Cool Chips Tutorial, held in conjunction with 32nd Annual International Symposium on Microarchitecture*, November 1999.

Kevin Skadron, Tarek Abdelzaher, and Mircea Stan, “Control-theoretic techniques and thermal-RC modeling for accurate and localized dynamic thermal management,” Tech. Rep. CS-2001-27, University of Virginia, 2001.

Vivek Tiwari, Deo Singh, Suresh Rajgopal, Gaurav Mehta, Rakesh Patel, and Franklin Baez, “Reducing power in high-performance microprocessors,” in *Design Automation Conference*, 1998, pp. 732–737.

Appendix A: DVS algorithm

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <string.h>
#include <sys/errno.h>
#include <stdarg.h>
#include <time.h>

// print debugging output
#define DEBUG 0

// set this to 1 for CPUFreq with sysfs interface, 0 for /proc/sys/cpu/*
#define SYSFS 1

// Maximum number of speed steps. Must be 2 or greater
#define MAX_SPEEDS 20

// To activate debug printings
#if DEBUG
# define errprintf(A...) fprintf(stderr,A)
#else
# define errprintf(A...)
#endif

#if SYSFS
# define MIN_SPEED_FILE "/sys/devices/sys/cpu0/scaling_min_freq"
# define MAX_SPEED_FILE "/sys/devices/sys/cpu0/scaling_max_freq"
# define CURRENT_SPEED_FILE "/sys/devices/sys/cpu0/scaling_setspeed"
# define GOVERNOR_FILE "/sys/devices/sys/cpu0/scaling_governor"
# define USERSPACE "userspace"
#else
# define MIN_SPEED_FILE "/proc/sys/cpu/0/speed-min"
# define MAX_SPEED_FILE "/proc/sys/cpu/0/speed-max"
```



```

#   define CURRENT_SPEED_FILE    "/proc/sys/cpu/0/speed"
#   define GOVERNOR_FILE        "/proc/cpufreq"
#   define USERSPACE            "0%0%100%userspace"
#endif

// defines what info we care about for each speed step
struct {
    unsigned khz;
    //unsigned volts;
    //unsigned fsb;
} speeds[MAX_SPEEDS + 1];

// display an error message and exit the program
void
die(bool system_error, const char *fmt, ...)
{
    fprintf(stderr, "Error: ");

    va_list ap;
    va_start(ap, fmt);          // get variable argument list passed
    vfprintf(stderr, fmt, ap); // display message passed on stderr
    va_end(ap);

    fprintf(stderr, "\n");
    if (system_error)
        fprintf(stderr, "Error: %s\n", strerror(errno));

    exit(1);
}

// read a line from a file
void
read_line(const char * filename, char * line, unsigned len)
{
    FILE * fp = fopen(filename, "r");
    if (!fp)
        die(true, "Could not open file for reading: %s", filename);
    if ( (!fgets(line, len, fp)) )
        die(true, "Could not read from file: %s", filename);
    fclose(fp);
}

// write a line to a file
void
write_line(const char * filename, const char *fmt, ...)
{
    FILE * fp = fopen(filename, "w+");

```

```

    if (!fp)
        die(true, "Could not open file for writing: %s", filename);
    va_list ap;
    va_start(ap, fmt);          // get variable argument list passed
    if (vfprintf(fp, fmt, ap) < 0)
        die(true, "Could not write to file: %s", filename);
    va_end(ap);
    fclose(fp);
}

// read an integer value from a file
unsigned
read_value(const char * filename)
{
    char line[256];
    read_line(filename, line, sizeof line);
    return atoi(line);
}

// get the current CPU speed
unsigned get_speed()
{
    return read_value(CURRENT_SPEED_FILE);
}

// set the current CPU speed
void set_speed(unsigned value)
{
    errprintf("Setting speed to: %uKHz\n", value);
    write_line(CURRENT_SPEED_FILE, "%u\n", value);
    usleep(10000); // CET - FIXME - Is this necessary for any CPUs?
}

void set_speed(unsigned current, unsigned target)
{
    if (current == target)
        return;
    int delta = (current > target) ? -1 : 1;
    do
    {
        current += delta;
        set_speed(speeds[current].khz);
    }
    while (current != target);
}

// lowest speed step

```

```

unsigned last_step;

// get the speed steps supported by the CPU
void
get_supported_speeds()
{
    unsigned min = read_value(MIN_SPEED_FILE);
    unsigned max = read_value(MAX_SPEED_FILE);
    const unsigned step = 25000;

    speeds[0].khz = max;
    unsigned current_speed = 0;
    for (unsigned current = max - step; current > min - step;
         current -= step)
    {
        set_speed(current);
        unsigned real = get_speed();
        if (real != speeds[current_speed].khz)
        {
            speeds[++current_speed].khz = real;
            if (current_speed + 1 == MAX_SPEEDS)
                break;
        }
    }
    speeds[current_speed + 1].khz = 0;

    // go back to maximum speed because program expects to start there
    set_speed(current_speed, 0);

    // the last step is the lowest found speed
    last_step = current_speed;

#ifdef DEBUG
    errprintf("Available speeds:\n");
    for (current_speed = 0; speeds[current_speed].khz; current_speed++)
        errprintf(" %2u: %9uKHz\n", current_speed,
                 speeds[current_speed].khz);
#endif
}

// gets the elapsed total time and elapsed idle time
//since it was last called
void
get_times(unsigned long & total_elapsed, unsigned long & idle_elapsed)
{
    char what[32];

```

```

unsigned long user_time, nice_time, system_time,
        idle_time, total_time;
static unsigned long last_total_time = 0, last_idle_time = 0;

bool found;
char line[256];
FILE * fp = fopen("/proc/stat", "r");
if (!fp)
    die(true, "Could not open /proc/stat for reading!");
while ( (found = fgets(line, sizeof line, fp)) )
{
    sscanf(line, "%s %lu %lu %lu %lu", what, &user_time, &nice_time,
            &system_time, &idle_time);
    if (!strcmp(what, "cpu0"))
        break;
}
fclose(fp);

if (!found)
    die(false, "Could not find entry for cpu0 in /proc/stat!?");

// count nice time as idle time
idle_time += nice_time;

total_time = user_time + system_time + idle_time;
total_elapsed = total_time - last_total_time;
last_total_time = total_time;
idle_elapsed = idle_time - last_idle_time;
last_idle_time = idle_time;

}

// resets the elapsed total time and elapsed idle time counters
void reset_times()
{
    unsigned long dummy1, dummy2;
    get_times(dummy1, dummy2);
}

// are we currently dynamically scaling the CPU or at min or max?
enum Mode { DYNAMIC, MIN, MAX } mode;

// if CPU idle/work ratio is below this,
//CPU will be set to next higher speed
float clock_up_idle=0.25;
// if CPU idle/work ratio is above this,

```

```

//CPU will be set to next lower speed
float clock_down_idle=0.75;

// handles the periodic check of idle and setting CPU speed
void
alarm_handler(int)
{
    static unsigned new_speed = 0;
    static unsigned current_speed = 0;

    // current state
    Mode state = mode;

    char line[256], *p;

    if (mode == DYNAMIC)
    {
        // figure out what our current speed should be
        switch(state)
        {
            case DYNAMIC:
            {
                unsigned long elapsed_time, idle_time;
                double         idle_ratio;

                // get the elapsed and idle times since we last checked
                get_times(elapsed_time, idle_time);

                if (elapsed_time > 0)
                {
                    idle_ratio = (double)idle_time / (double)elapsed_time;

                    if (idle_ratio <= clock_up_idle && new_speed != 0)
                        new_speed--;
                    else if (idle_ratio >= clock_down_idle
                            && speeds[new_speed + 1].khz != 0)
                        new_speed++;

                    errprintf("Idle ratio: %.2f   %4dKHz\n",
                            idle_ratio, speeds[new_speed].khz);
                }
                break;
            }

            case MIN:
                new_speed = last_step;
        }
    }
}

```

```

        break;

    case MAX:
        new_speed = 0;
        break;
}

// if the last set speed is not what it currently should be,
//set it
if (current_speed != new_speed)
{
    errprintf("current speed: %uKHz      new speed: %uKHz\n",
              speeds[current_speed].khz, speeds[new_speed].khz);
    set_speed(current_speed, new_speed);
    current_speed = new_speed;
}
}

// handles the USR1 signal (stay at maximum performance)
void
usr1_handler(int)
{
    mode = MAX;
}

// handles the USR2 signal (stay at minimum performance)
void
usr2_handler(int)
{
    mode = MIN;
}

// handles the HUP signal (dynamically scale performance)
void
hup_handler(int)
{
    reset_times();
    mode = DYNAMIC;
}

long
get_int(const char * s)
{
    char * end;
    long r = strtol(s, &end, 10);
    if (*end != '\0')
        die(false, "Not an integer: %s", s);
}

```

```

    if (errno == ERANGE)
        die(false, "Number is out of range: %s", s);

    return r;
}

float
get_float(const char * s)
{
    char * end;
    float r = (float)strtod(s, &end);
    if (*end != '\0')
        die(false, "Not a floating point number: %s", s);
    if (errno == ERANGE)
        die(false, "Number is out of range: %s", s);

    return r;
}

int
main(unsigned argc, char * argv[])
{
    unsigned interval = 20; // 2 seconds
    bool daemonize = false;

    // parse argv
    for(unsigned i = 1; i < argc; i++)
    {
        if(!strcmp(argv[i], "-d"))
            daemonize = true;
        else if(!strcmp(argv[i], "-i"))
        {
            if (argc <= i + 1)
                die(false,
                    "The -i option must be followed by an interval
in tenths of a second\n");

            interval = get_int(argv[++i]);
            errprintf("Interval is %u\n", interval);
        }
        else if(!strcmp(argv[i], "-p"))
        {
            if (argc <= i + 3)
                die(false,
                    "The -p option must be followed by 3 floats\n");

```

```

        clock_up_idle_fast = get_float(argv[++i]);
        clock_up_idle = get_float(argv[++i]);
        clock_down_idle = get_float(argv[++i]);
        errprintf("Triggers are %f %f %f\n",
            clock_up_idle_fast, clock_up_idle, clock_down_idle);
    }
else if (!strcmp(argv[i], "-a"))
{
    if (argc <= i + 1)
        die(false,
            "The -a option must be followed by a filename\n");

    ac_filename = argv[++i];
}
}

// use the userspace governor
write_line(GOVERNOR_FILE, "%s\n", USERSPACE);

if (access(CURRENT_SPEED_FILE, W_OK) < 0)
    die(true, "Cannot write to speed control file: %s",
        CURRENT_SPEED_FILE);

// run in background if requested
if (daemonize)
    daemon(0, 0);

// set up signal handling
struct sigaction signal_action;

sigemptyset(&signal_action.sa_mask);
sigaddset(&signal_action.sa_mask, SIGALRM);
sigaddset(&signal_action.sa_mask, SIGUSR1);
sigaddset(&signal_action.sa_mask, SIGUSR2);
sigaddset(&signal_action.sa_mask, SIGHUP);
signal_action.sa_flags = 0;

// set the SIGALRM handler to our function
signal_action.sa_handler = alarm_handler;
sigaction(SIGALRM, &signal_action, 0);

// set the SIGUSR1 handler to our function
signal_action.sa_handler = usr1_handler;
sigaction(SIGUSR1, &signal_action, 0);

// set the SIGUSR2 handler to our function
signal_action.sa_handler = usr2_handler;

```



```
sigaction(SIGUSR2, &signal_action, 0);

// set the HUP handler to our function
signal_action.sa_handler = hup_handler;
sigaction(SIGHUP, &signal_action, 0);

// reset the speed steps
get_supported_speeds();
// reset the time counters
reset_times();

// we dynamically scale speed to start
mode = DYNAMIC;

// main loop
while (1)
{
    raise(SIGALRM);
    usleep(interval * 100000L);
}

// never reached
return 0;
}
```