



NORTHWESTERN UNIVERSITY

Electrical Engineering and Computer Science Department

Technical Report
NWU-EECS-10-09
August 20, 2010

The End User in Computer Architecture and Systems Research

Alex Shye

Abstract

The ultimate goal of a computer design is to satisfy the end user. However, the design and optimization of computer architectures have largely left the user out of the loop. In this dissertation, I make the case that with modern computer architectures it is becoming increasingly important to take the end user into account. I then propose three specific aspects of the end user that should be explored when incorporating the end user into loop; (1) user perception, (2) user state, and (3) user activity.

First, I show that that computer architects should study the end user's perception of performance relative to actual hardware performance. User studies show that for satisfaction across different users. This variation represents opportunity for optimizing computer architectures subject to individual user satisfaction. Second, I make the case for measuring user state via empathic input devices, input devices providing a computer with information about user state. I demonstrate that three example empathic input devices (eye tracking, a galvanic skin response sensor, and force sensors) can be useful for understanding changes in user satisfaction for driving power optimizations. Third, I show that computer architects should begin studying the activity of the end user as an important part of the workload. I study real user activity on Android G1 mobile phones and to show that it can be important in characterizing power consumption, and developing new power optimizations.

Overall, this work points towards a new approach to computer architecture and systems research that incorporates the end user into the loop. The findings show that if we place the end user into the design and optimization process, we can significantly improve the efficiency of current computer architectures and systems, while maintaining or even improving individual user satisfaction at the same time.

Keywords: Human Factors, Power Management, Computer Architecture, Mobile Computing

NORTHWESTERN UNIVERSITY

The End User in Computer Architecture and Systems Research

A DISSERTATION

SUBMITTED TO THE GRADUATE SCHOOL
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree

DOCTOR OF PHILOSOPHY

Field of Electrical and Computer Engineering

By

Alex Shye

EVANSTON, ILLINOIS

June 2010

© Copyright by Alex Shye 2010

All Rights Reserved

ABSTRACT

The End User in Computer Architecture and Systems Research

Alex Shye

The ultimate goal of a computer design is to satisfy the end user. However, the design and optimization of computer architectures have largely left the user out of the loop. In this dissertation, I make the case that with modern computer architectures it is becoming increasingly important to take the end user into account. I then propose three specific aspects of the end user that should be explored when incorporating the end user into loop; (1) user perception, (2) user state, and (3) user activity.

First, I show that that computer architects should study the end user's perception of performance relative to actual hardware performance. User studies show that for modern interactive and multimedia applications, there exists a significant variation in user satisfaction across different users. This variation represents opportunity for optimizing computer architectures subject to individual user satisfaction. Second, I make the case for measuring user state via empathic input devices, input devices providing a computer with information about user state. I demonstrate that three example empathic input devices (eye tracking, a galvanic skin response sensor, and force sensors) can be useful

for understanding changes in user satisfaction for driving power optimizations. Third, I show that computer architects should begin studying the activity of the end user as an important part of the workload. I study real user activity on Android G1 mobile phones and to show that it can be important in characterizing power consumption, and developing new power optimizations.

Overall, this work points towards a new approach to computer architecture and systems research that incorporates the end user into the loop. The findings show that if we place the end user into the design and optimization process, we can significantly improve the efficiency of current computer architectures and systems, while maintaining or even improving individual user satisfaction at the same time.

Acknowledgements

They say it takes a whole village to raise a child. It feels like it has taken *several* whole villages to raise this child. I am very fortunate to have many people who have played a significant role in my life and career. They have offered great advice (although I sometimes don't listen!), sweat in the lab with me (including a few too many all-nighters), consoled me during tough times (getting a Ph.D. is no cakewalk), and challenged me to be a better person and researcher (I'm getting there.. slowly but surely :).

First and foremost, I thank my dissertation advisor, Prof. Gokhan Memik, for his support ¹ during my time at Northwestern University. He has taught me more than I can describe here, but I am most thankful for his constant encouragement to follow my interests, even when it takes being courageous with research topics. I thank Prof. Peter Dinda and Prof. Robert Dick for their great guidance and feedback. They both played a large role in my positive experience at Northwestern, and in shaping the work in this dissertation. In addition, I thank Prof. Seda Memik, Prof. Russ Joseph, and Prof. Nikos Hardavellas, Prof. Bryan Pardo, Prof. Darren Gergle for their assistance, encouragement, feedback, and advice. I thank Prof. Daniel A. Connors for giving me my first crack at research many years ago. I would not be where I am right now without his support and

¹Of course, this includes financial support. This work is in part supported by DOE Awards DE-FG02-05ER25691 and DE-AC05-00OR22725 (via ORNL), NSF Awards CNS-0720691, CNS-0721978, CNS-0715612, CNS-0551639, CNS-0347941, CCF-0541337, CCF-0444405, CCF-0747201, IIS-0536994, IIS-0613568, ANI-0093221, ANI-0301108, and EIA-0224449, by SRC award 2007-HJ-1593, by Wissner-Slivka Chair funds, and by gifts from Symantec, Dell, and VMware.

guidance. I also thank Prof. Manish Vachharajani for his valuable feedback and advice over the years.

I am lucky to have several hosts and colleagues within industry that have provided me with valuable industry research experience. I thank Evelyn Duesterwald, Calin Cascaval, Robert Wisniewski, and Peter Sweeney for my time at IBM Research. I thank John Pieper for mentoring me at Intel, and Brad Chen for hosting me at Google. And, I give a big thanks to the DynOpt group (Mark Herdeg, Anton Chernoff, Joyce Spencer, Tony Tye, Michael Bedy, Roland Ouellette, Rick Gorton, Joe Martin, and Walter Carrell) for a wonderful 2-year co-op at AMD.

I sincerely thank all of my collaborators: Arindam Mallik, Berkin Ozisikyilmaz, Yan Pan, J. Scott Miller, Benjamin Scholbrock, Lei Yang, Xi Chen, and Bin Lin at Northwestern University; Tipp Moseley, Vijay Janapa Reddi, Matthew Iyer, Joseph Blomstedt, Joshua Kihm, Alex Settle, Dan Fay, and Dave Hodgdon at the University of Colorado. We worked hard, sweated it out in lab, pulled all-nighters, and, most importantly, had a great deal of fun in the process. All of the work would not have been possible without your contributions. In addition, I would like to thank the rest of the Microarchitecture Research Lab and the related labs at Northwestern for being a sounding board with research, and tolerating my antics at Northwestern.

I thank Arty Plengsirivat for her companionship during the entire Ph.D. process – celebrating with me during the good times, consoling me during the tough times, and balancing out my life to keep me sane.

Last, but certainly not least, I thank my family; my father Ken Shye, my mother Shou Shye, and my brother Michael Shye. Thank you for believing in me all of these years; your constant support means the world to me.

Table of Contents

ABSTRACT	3
Acknowledgements	5
List of Tables	11
List of Figures	12
Chapter 1. Introduction	16
1.1. The Forgotten End User	17
1.2. Why Care About the User Now?	20
1.3. Putting the End User into the Loop	23
1.4. For Reference	27
Chapter 2. User Perception: Leveraging Individual User Satisfaction	28
2.1. Hardware Performance Counters	31
2.2. User Study Setup	32
2.3. Correlation Between User Satisfaction and Hardware Counters	34
2.4. Leveraging User Variation with Predictive Power Management	38
2.5. Experimental Results	45
2.6. Summary	52
2.7. Appendix: Correlating HPC Metrics and User Satisfaction	53

Chapter 3. User Physiological Traits: Implicitly Learning User Satisfaction	55
3.1. Empathic Input Devices	58
3.2. User Study Setup	63
3.3. Correlating Human Physiological Traits with User Satisfaction	64
3.4. Using Physiological Traits for Dynamic Voltage and Frequency Scaling	70
3.5. Experimental Results	76
3.6. Summary	82
3.7. Raw Data from Motivational User Study	83
Chapter 4. User Activity: Studying User Behavior to Drive Optimization	85
4.1. Experimental Setup	88
4.2. Power Estimation Model	91
4.3. Studying the User for Guiding Optimization	102
4.4. User-Aware Optimization	108
4.5. Experimental Results	112
4.6. Summary	118
Chapter 5. Related Work	119
5.1. The Empathic Systems Project	119
5.2. Other User-Related Work	120
5.3. Measuring the End User	122
5.4. Power Modeling	122
5.5. Dynamic Voltage and Frequency Scaling	123
5.6. Screen Optimizations	124

	10
Chapter 6. Conclusion	125
References	127
Vita	135

List of Tables

2.1	Hardware counters used in experiments.	32
2.2	User trend categorization, the number of users in each category for different applications.	37
2.3	Correlation of hardware counters and user satisfaction.	54
3.1	Outcomes of manually comparing t-test results and the user satisfaction ratings. Success means that the t-test outcome matches the user rating. False negatives occur when the t-test falsely predicts a difference and false positives occur when the t-test falsely predicts similarity with the highest frequency.	67
4.1	Parameters used for linear regression in our power estimation model.	92

List of Figures

1.1	The (a) traditional computing stack, and the (b) computing stack including the end user.	18
1.2	The evolution of computer architectures. Each generation through the years is because more “personal”, increasing the importance of incorporating studies of the end user.	22
1.3	The end user in the computing loop.	23
2.1	Framework of the predictive user-aware power management.	38
2.2	Figure 2. Frequency traces using <i>i</i> DVFS and Windows power management schemes for (a) Java Game and (b) Video.	43
2.3	Windows DVFS algorithm.	46
2.4	Reported user satisfaction for the (a) Web Browsing, (b) Game, and (c) Video applications.	48
2.5	System power measurement setup.	52
2.6	Figure 6. Improvement in energy consumption, user satisfaction, and energy-satisfaction product for the Shockwave application.	53

- 3.1 The biometric sensors used in our experiments: (a) an eye tracker, (b) a custom-made galvanic skin response sensor, and (c) force sensors attached to the arrow keys on the keyboard. 59
- 3.2 GSR traces of a user that capture (a) the long-term change in the GSR while a user is resting, and (b) the short-term effects when playing the Need for Speed game. The existence of the long-term effect motivates the use of the delta GSR metric for measuring user arousal. 60
- 3.3 (a) Mean pupil movement, (b) maximum arrow force, and (c) maximum delta GSR for the same 20 seconds of game play at a good performance level, and at a bad performance level. Mean pupil movement and maximum arrow force significantly decrease. Maximum delta GSR has more variation across users indicating different responses to a drop in performance. 65
- 3.4 Averages of the three best individual sensor metrics and the user satisfaction ratings across all 20 users. The three sensor metrics have a very strong correlation with the reported user rating. 68
- 3.5 The average confidence provided by the t-test between a frequency and the highest frequency across all 20 users and all sensor metrics. A high confidence indicates a difference. As frequency difference increases, the sensor metrics differentiate better, except for the lowest frequency. 69
- 3.6 Trace of sensor metrics and the frequency during the training phase of the PTP algorithm. When sensor readings are compared for 1.2Ghz and

2.2Ghz, the majority of the sensors result in a high t-test, indicating that the user's state changes. As the algorithm adjusts to test 1.6 GHz, the physiological traits show less change. PTP chooses 1.6Ghz for the rest of the experiment

72

3.7 Frequency that *a*PTP and *c*PTP settle on for the Need for Speed, Tetris, and Word applications. *c*PTP for Word is omitted because it results in very little change in power savings and user satisfaction.

77

3.8 User satisfaction and power consumption for the Need for Speed, Tetris, and Word applications. The left two bars per cluster show the user satisfaction for *a*PTP and the Adaptive DVFS schemes. The right bar in each cluster shows the total system power savings.

80

3.9 User satisfaction and power consumption of *c*PTP for the Need for Speed and Tetris applications. Word is not included because power savings and user satisfaction levels are nearly identical to *a*PTP. The left two bars per cluster show the user satisfaction of *c*PTP and the Adaptive DVFS schemes. The right bar in each cluster shows the total system power savings. Using *c*PTP, we trade-off a decreased power savings with improving user satisfaction when compared to *a*PTP.

81

3.10 Physiological traits and user satisfaction when randomly changing to multiple frequencies at different points in Need for Speed.

84

4.1 High-level overview of the target mobile architecture.

88

4.2	Error of logger when building the power estimation model on one ADP1 and validating with logs from another ADP1 device.	98
4.3	Cumulative distribution of power estimation error.	99
4.4	Cumulative total energy error.	100
4.5	Power consumption timeline.	100
4.6	Power consumption breakdown for traces that stress specific hardware units.	101
4.7	Power consumption breakdown from real user traces.	104
4.8	Screen durations based upon user activity.	107
4.9	Total system power savings for each of the optimizations as estimated by our power model.	113
4.10	Reported user satisfaction for the (a) Web Browsing, (b) Game, and (c) Video applications.	115

CHAPTER 1

Introduction

“Focus on the user and all else will follow.”

Google, #1 principle in Google Philosophy

The ultimate goal of a computer design is to satisfy the end user. This statement may sound simple – perhaps even obvious – but it is a very powerful statement. It means that after all the work we put into design and optimization, we can evaluate a computer system by putting it into the hands of the end user. If the user is happy with it, we have done a good job. If not, perhaps we can do better.

This statement regarding the importance of the end user is not a new idea in the computer industry. Google understands this statement. As quoted in the beginning of this introduction, the number one principle in their corporate philosophy is to focus on the end user [47]. This focus is clear in their products, and pervades the entire user experience, from their crisp home page, to the responsiveness and quality of their web search. Apple understands this statement. Apple’s focus on the user for designing simple and intuitive user interfaces has revolutionized the computer industry. Engineers and researchers in several other computing-related fields understand this statement. For example, at the application level, multimedia experts study the perceptual quality of a media [28, 29, 61, 85, 90], and human-computer interaction researchers develop applications for improving the human condition [24, 25, 26].

However, despite the importance of the end user, computer architects and systems designers have largely ignored the end user. The work in this dissertation argues that this should not be the case. User experience matters more than ever. Decisions at the architecture- and systems-level impact the user experience, and must be made with end user in mind. If we take the user into account during the architectural design process, we can improve the efficiency and performance of computer architectures, while maintaining, or even improving, user satisfaction.

1.1. The Forgotten End User

The design and optimization of computer architectures has typically left the end user out of the loop. It is not difficult to understand why this is the case.

Where would the end user fit with respect to computer design? Traditionally, the term “computer” refers to a programmable machine. The Merriam-Webster Online Dictionary defines a computer as, “a programmable, usually electronic device, that can store, retrieve, and process data”. Other definitions do not stray too far from this general idea. Thus, computer design has typically focused on three tasks: (1) specifying a set of instructions for data access/manipulation, (2) designing the circuits/hardware for implementing the instructions, and (3) developing systems software and tools to provide an environment for running different mixes of instructions. None of these tasks involve the end user.

Where would the end user fit with respect to optimizing computers? At the core, optimization involves (1) choosing a performance metric, and (2) an iterative loop of doing a baseline performance measurement, implementing/tuning an optimization, doing

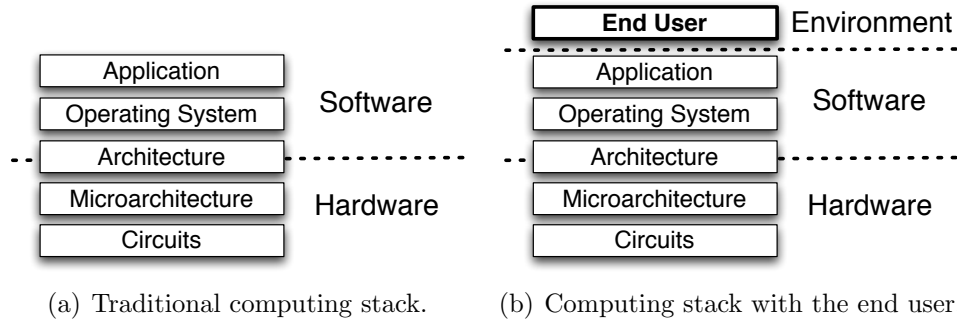


Figure 1.1. The (a) traditional computing stack, and the (b) computing stack including the end user.

another performance measurement, and crossing your fingers for a performance improvement. There are many traditional performance metrics typically used: instructions per second, transactions per second, energy consumption, soft error rate, failures in time, average system up time, etc. However, none of the traditional performance metrics typically involve the end user.

Of course, there *is* an end user involved with a computer. After all, we did invent and create computers to do work for us. So, where does the end user fit into the picture? And, where does the end user fit into computing research?

To answer these questions, it may be helpful if we observe the textbook computing stack, as shown in Figure 1.1(a). This traditional computing stack includes (1) the circuits and microarchitecture at the hardware level, (2) the architecture for specifying the hardware-software interface, and (3) the software layers, with the operating system (OS) and application. Note that this computing stack only accounts for hardware and software. Clearly, the end user is neither hardware or software. Thus, there is no room for the end user in this picture. However, computation does not occur in a vacuum. If we expand our scope, we can add the *environment* to the top of the computing stack, to account for all

of the characteristics and factors of the environment that may interact with computation. The end user naturally fits into the environment at the top of the computing stack, as shown in Figure 1.1(b) ¹.

Now that we have the end user in the computing stack, where is the end user with respect to computing research? If we look at the interaction between layers in the computing stack, we arrive at an explanation for the state of the end user in current computing research. The design of a computer can be very complex. To manage complexity, we use abstraction by specifying an interface for accessing low-level details. For example, logical operators, such as AND and OR gates, specify the interface between circuits and the microarchitecture. The instruction set architecture is the interface between the microarchitecture and software. System calls specify the interface between the operating system and the application. Abstraction allows engineers to design to interfaces, without needing to know the gory details under the hood. It also means that the majority of research lies within a single layer, or spans adjacent layers in the stack that interact with each other.

The layers of the computing stack show us that with respect to the end user, it is most natural for computer researchers to study the interactions of the end user with the application ². There are many important questions in this area. How should end users interact with applications? Which hardware devices are most natural and intuitive for users? How can application interfaces be designed to improve usability? These questions, and many more, have spawned an entire field of research, human-computer interaction (HCI).

¹There may be many other factors in the environment (e.g., perhaps the energy source, room temperature, etc.). In this dissertation, the main focus is the end user, and thus, we do not discuss other potential aspects of the environment.

²Studying the end user within its own layer is most likely best left for psychologists.

The computing stack also shows us what low-level hardware and software designers usually think of as the “end user” – the application software. We usually think of the application as interacting with the hardware and systems software. Thus, it becomes our proxy for the end user. Instead of studying the user, we study the behavior of an application. To model a “average” user, we use mixes of representative applications as benchmarks. A look at most any modern architecture or systems research paper will show the use of benchmarks for evaluating a proposed technique.

In summary, although a computer is ultimately designed to satisfy the end user, there is currently a disconnect between the design of the low-level hardware/software and the end user. Most of the user-related research occurs at the application level, or in the interaction between the application and the end user. The levels in the computing stack have largely ignored the end user. Instead, we have abstracted away the end user. All that remains is a representative set of applications.

1.2. Why Care About the User Now?

This dissertation makes the case that it is time to expand the role of the end user in computer architecture and systems research. In particular, three trends are converging that increase the role of the end user in modern computer systems:

- (1) **The importance of user experience.** Batch applications are not the sole workloads for most architectures. Modern multimedia applications, video games, web browsers, and server-side applications interact directly with the end user. Applications on mobile devices are inherently interactive. Although traditional

metrics (i.e., instructions per second) may be important in evaluating these machines and applications, the most important thing is whether the user is satisfied or not. To ensure a good user experience, we must move beyond traditional metrics, and develop user-related performance evaluation techniques for understanding the impact of architectural and optimization decisions on user satisfaction.

- (2) **Architectural trade-offs are directly exposed to the end user.** All aspects of an architecture, including performance, power consumption, temperature, and lifetime reliability, are now directly exposed to the end user. For example, the end user can determine when the performance of a computer is satisfactory, but is also painfully aware when the operating temperature of a laptop is too high, or when the battery life is surprisingly short. To balance these tradeoffs effectively, architects must take the end user into account when tuning and optimizing architectures.
- (3) **The end user drives the workload.** The first step to optimization is often to understand the workload. As modern applications become increasingly interactive, their workload will become increasingly dependent upon the actions and behavior of the end user. Treating the user as an important part of the workload may reveal new trends, patterns, or properties that can be leveraged for optimization.

Underlying these trends is a continual shift towards delivering the end user an ever-more personal computer experience. In the past few decades, we have seen a dramatic evolution in computer architectures, as shown in Figure 1.2. We have seen a giant leap

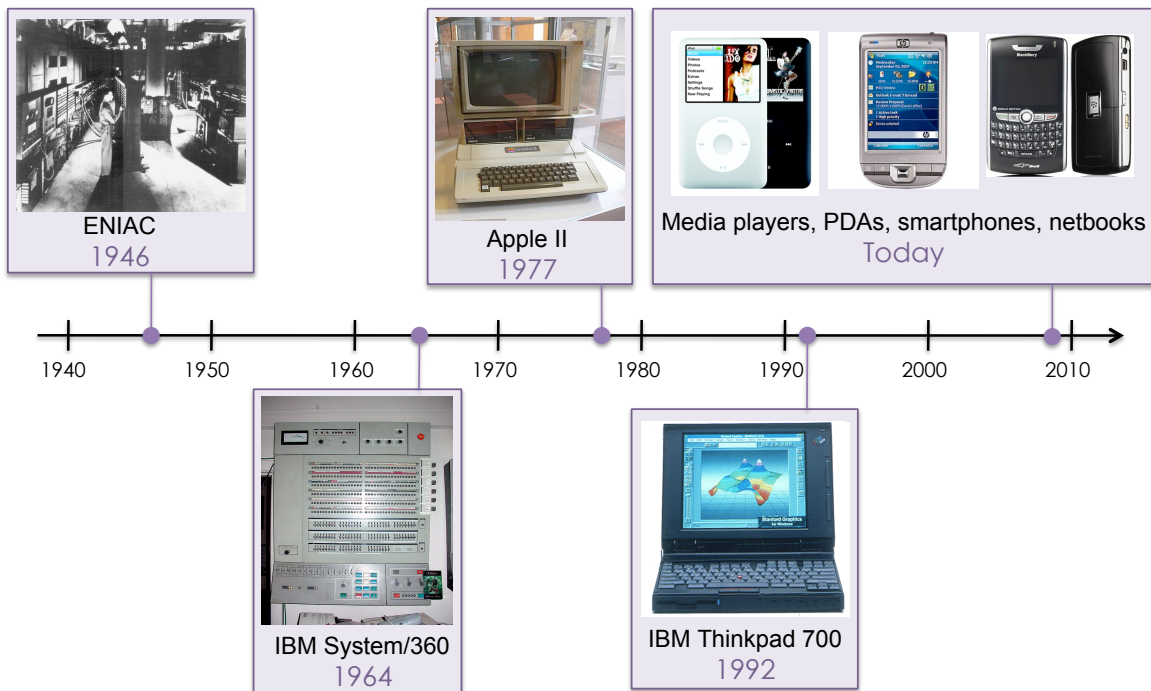


Figure 1.2. The evolution of computer architectures. Each generation through the years is because more “personal”, increasing the importance of incorporating studies of the end user.

from the *supercomputing* era (e.g., room-sized vacuum-tubed-based computers, mainframes) to the *personal* computing era (e.g., desktop computers). We are currently in the midst of another significant leap to the *portable* computing era (e.g., netbooks, PDAs, smartphones). Technology has advanced to a point where computation and communication can be effectively integrated into small handheld devices. Users are integrating a mix of these mobile devices into their daily lives, making these devices their source of on-the-go computation, hub of communication, and portal to the growing wealth of information on the web. We can expect the personalization of computers to continue beyond the portable computing era, into what many dub the *pervasive* computing era, where computers pervade all aspects of our daily lives, including our utilities, our clothes,

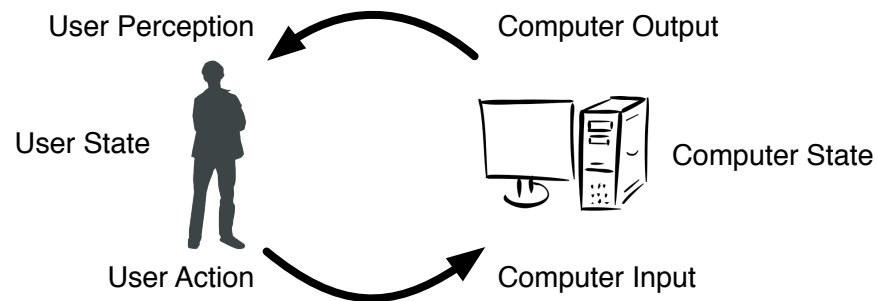


Figure 1.3. The end user in the computing loop.

and even our brains. As we progress further into the portable computer era, and reach into the pervasive computing era, the three previously mentioned trends will continue, and grow in importance.

1.3. Putting the End User into the Loop

We begin incorporating the end user by taking a high-level view of the traditional interaction between the end user and the computer (shown in Figure 1.3).

The interaction from the side of the computer is familiar. The computer receives input, e.g., a keystroke or mouse movement. It then executes instructions based upon the input and alters computer state, e.g., the contents of memory and disk. Finally, any computer state of interest to the end user may be output, e.g., through the display or speakers.

This dissertation focuses on the other side of this interaction. It focuses on the human side of the human-computer interaction, and what it means to the decisions that should be made at the architecture- and system-level. By observing the interaction from the perspective of the end user, we find three key points of interest in the computer design process: (1) user perception, (2) user state, and (3) user activity. These three points are

the main focus of this dissertation. They are described briefly below, and in detail in each of the following chapters.

1.3.1. User Perception

The first point of interest is the end user's perception of computer behavior/performance. Note that it is not the actual input we are interested in. We distinguish the actual stimulus from the perception of the stimulus. This is an important distinction. When focusing on the experience of the user, it is really the perception of the stimulus that is most important, not necessarily the stimulus itself.

Chapter 2 studies user perception of computer performance relative to raw hardware performance. We perform real user studies to study *user satisfaction* (a verbal user rating of perceived computer performance) relative to raw hardware performance. We present three main contributions.

- (1) First, I show that the relationship between user satisfaction and hardware performance is often a complex non-linear relationship that is application dependent, and more importantly, user dependent. Our results show that *there is no average user*. Instead, there exists a variation in perceived performance across individual users. We refer to this variation as *user variation*.
- (2) Second, I unveil a relationship between hardware performance counters on modern microprocessors. We show that we can learn this relationship by mapping hardware counter values to user satisfaction, and then use this mapping as a proxy for predicting user satisfaction.

- (3) Third, I show that these hardware-satisfaction models can be leveraged to optimize subject to user variation. I demonstrate Individualized Dynamic Voltage and Frequency Scaling (*iDVFS*) a system that uses a per-user model to drive dynamic voltage and frequency scaling (DVFS) on CPUs based upon the preferences of the individual user.

1.3.2. User State

The second point in the human-computer interaction we are interested in is user state. We use the term ‘user state’ broadly to account for all user-related factors that may represent the state of the user, including bodily position, emotions, intentions, physiological traits, etc. With respect to user state, we are particularly interested in any user state that may indicate whether the user is satisfied with decisions at the architecture- or systems-level.

Chapter 3 presents a study of leveraging user state for optimizing computer architectures. We make three main contributions in this chapter.

- (1) I propose new *empathic input devices* to measure human physiological traits and provide the computer with information on user state. Specifically, I propose using eye trackers, a galvanic skin response sensor, and force sensors as potential empathic input devices.
- (2) I present two user studies to show evidence that these empathic input devices can be used to reason about changes in user satisfaction.
- (3) I augment an existing DVFS scheme to make decisions based upon human physiological traits, and demonstrate success at improving energy efficiency for interactive applications.

1.3.3. User Activity

The third point in current human-computer interaction we are interested in is essentially user output. I treat user output as the activity of the end user, e.g., keystrokes, mouse events, turning the computer on, etc. As computers become increasingly personal, and pervade our society, the activity of individual users, as well as users in general, will become an increasingly important characteristic of the computer workload. Thus, studying trends, patterns, and properties of user activity may be useful in characterizing the workload, as well as driving future optimizations.

Chapter 4 studies user activity for perhaps the most personal of current computer architectures, mobile smartphones. I show that studying user activity on a specific smartphone, the Android G1 phone, can be instrumental in understanding the power consumption of smartphones in the wild, and in unveiling new user-activity-related properties for developing power optimizations. I make the following contributions:

- (1) I present a logger application for collecting data from remote smartphones, and a validated power model driven to traces from the logger applications for estimating the power consumption of remote smartphones.
- (2) I demonstrate that user activity plays a significant role in the activity, and power consumption breakdown, of mobile smartphones. Along the way, I show that the CPU and screen consume significant power in mobile smartphones and are good candidates for power optimization.
- (3) I observe that screen on time is dominated by a small number of long screen durations and propose an optimization for long screen intervals that leverage change blindness, a phenomenon where humans are often unable to perceive

changes in a given stimulus. We show that *change-blindness-based* optimizations may show promise, especially for the screen.

1.4. For Reference

For reference, the main contributions presented in this dissertation have previously appeared in the following conference publications:

- Alex Shye, Berkin Ozisikyilmaz, Arindam Mallik, Gokhan Memik, Peter A. Dinda, Robert P. Dick, and Alok N. Choudhary. **Learning and Leveraging the Relationship Between Architecture-Level Measurements and Individual User Satisfaction.** *In proceedings of the International Symposium on Computer Architecture*, June 2008. [91] (Chapter 2)
- Alex Shye, Yan Pan, Benjamin Scholbrock, J. Scott Miller, Gokhan Memik, Peter A. Dinda, and Robert P. Dick. **Power to the People: Leveraging Human Physiological Traits to Control Microprocessor Frequency.** *In proceedings of the International Symposium on Microarchitecture*, December 2008. [92] (Chapter 3)
- Alex Shye, Benjamin Scholbrock, and Gokhan Memik. **Into the Wild: Studying Real User Activity Patterns to Guide Power Optimizations for Mobile Architectures.** *In proceedings of the International Symposium on Microarchitecture*, December 2009. [93] (Chapter 4)

As the citations show, the work in this presentation is a collaborative work with several co-authors who have made critical contributions to the work. Thus, any mentions of “we” in the subsequent chapters refer to myself as well as my co-authors.

CHAPTER 2

User Perception: Leveraging Individual User Satisfaction

Any architectural optimization (performance, power, reliability, security, etc.) ultimately aims at satisfying the end-user. However, understanding the happiness of the user during the run of an application is complicated. Although it may be possible to query the user frequently, such explicit interaction will annoy most users. Therefore, it would be beneficial to estimate user satisfaction using implicit metrics. Traditionally, computer architects have used implicit metrics such as instructions retired per second (IPS), processor frequency, or the instructions per cycle (IPC) as optimization objectives. The assumption behind these metrics is that they relate in a simple way to the satisfaction of the user. When two systems are compared, it is assumed, for example, that the system providing a higher IPS will result in higher user satisfaction. For some application domains, this assumption is generally correct. For example, the execution time of a long running batch application is largely determined by the IPS of the processor. Hence, increasing IPS will result in an increase in user satisfaction. However, in this chapter we show that the relationship between hardware performance and user satisfaction is complex for interactive applications and an increase in a metric like IPS does not necessarily result in an increase in user satisfaction. More importantly, we show that the relationship between hardware performance and user satisfaction is highly user-dependent. Hence, we explore the feasibility of estimating individual user satisfaction from hardware metrics, develop accurate nonlinear models to do so, and use these models for run-time power management.

Driving architectural decisions from estimates of user satisfaction has several advantages. First, user satisfaction is highly user-dependent. This observation is not surprising. For example, an expert gamer will likely demand considerably more computational power than a novice user. In addition, each user has a certain “taste”; for example, some users prefer to prolong battery life, while others prefer higher performance. If we know the individual users satisfaction with minimal perturbation of program execution, we will be able to provide a better experience for the user. Second, when a system optimizes for user satisfaction, it will automatically customize for each application. Specifically, a system that knows the users satisfaction with a given application will provide the necessary performance to the user. For interactive applications, this may result in significant advantages such as power savings or increased lifetime reliability. For example, one of our target applications exhibits no observable change in performance when the frequency of the processor is set to its lowest level. In this case, our system drastically reduces the power consumption compared to traditional approaches without sacrificing user satisfaction.

Ultimately, our goal is to map microarchitectural information to user satisfaction. Such a map can then be used to understand how changes in microarchitectural metrics affect user satisfaction. Modern microprocessors contain integrated *hardware performance counters* (HPCs) that count architectural events (e.g., cache misses) as well as a variety of events related to memory and operating system behavior [4, 54, 55]. In this work, we aim at finding a mapping from the HPC readings to user satisfaction. We first show that there is a strong correlation between the HPCs and user satisfaction. However, the relationship between the two is often non-linear and user-dependent.

A good estimate of user satisfaction derived from microarchitectural metrics can be used to minimize power consumption while keeping users satisfied. Although utilizing user satisfaction in making architectural decisions can be employed in many scenarios, in this work, we focus on *dynamic voltage and frequency scaling* (DVFS) [19], which is one of the most commonly used power reduction techniques in modern processors. DVFS make decisions online to change microprocessor frequency and voltage according to processing needs. Existing DVFS techniques in high-performance processors select an operating point (CPU frequency and voltage) based on the utilization of the processor. Like many other architectural optimizations, DVFS is pessimistic about user satisfaction and assumes that the maximum processor frequency is necessary for every process that has a high CPU utilization. We show that incorporating user satisfaction into the decision making process can improve the power reduction yielded by DVFS. Specifically, our contributions in this work follow:

- We unveil a strong relationship between HPCs and user satisfaction for interactive applications;
- We show that this relationship is often non-linear, complex, and highly user-dependent;
- We show that individual user satisfaction can be accurately predicted using neural network models;
- We design Individualized Dynamic Voltage and Frequency Scaling (iDVFS), which employs user satisfaction prediction in making decisions about the frequency of the processor; and

- We implement and evaluate iDVFS on Windows with user studies that show it reduces power consumption compared to Window DVFS.

The chapter is organized as follows. In Section 2.1, we give an introduction to hardware counters. Section 2.2 describes our user study process. Section 2.3 presents results showing the relationship between user satisfaction and hardware counters. The predictive user-aware power management scheme is described in Section 2.4. Section 2.5 presents the results obtained from user studies. Section 2.6 summarizes our contributions.

2.1. Hardware Performance Counters

Modern microprocessors include integrated hardware performance counters (HPC) for non-intrusive monitoring of a variety of processor and memory system events [4, 54, 55]. HPCs provide low-overhead access to a wealth of detailed performance information related to CPU’s functional units, caches, main memory, etc. Even though this information is generally statistical in nature, it does provide a window into certain behaviors that are otherwise impractical to observe. For instance, these events include various counts of instructions, cache activity, branch mispredictions, memory coherence operations, and functional unit usage. Several tools and microprocessors have extended this functionality beyond simple event counting. For example, Intel’s Itanium processors [55] have features that allow monitoring specific events based on an instruction or data address range, a specific instruction opcode, or execution at specific privilege levels.

Current microprocessors support a limited number of HPCs. For example, the IA-64 architectures only support counting four events at a time [55]. In our experiments, we use the Pentium M processor which only supports two counters at a time. As a result,

PAPI counter	Description
PAPI_TOT_INS	Instructions issued
PAPI_RES_STL	Cycles stalled on any resource
PAPI_TOT_CYC	Total cycles
PAPIL2_TCM	Level 2 cache misses
PAPLBTAC_M	Branch target address cache misses
PAPI_BR_MSP	Conditional branch instructions mispredicted
PAPI_HW_INT	Hardware interrupts
PAPIL1_DCA	Level 1 data cache accesses
PAPIL1_ICA	Level 1 instruction cache accesses

Table 2.1. Hardware counters used in experiments.

it is not possible to collect all hardware information simultaneously. One workaround is to time multiplex sets of counters and then scale the values appropriately. Azimi, Stum, and Wisniewski [11] show that time multiplexing up to 10 sets of counters provides statistically significant counter values. Despite this limitation, the low-overhead access to low-level architectural information provided by HPCs is very useful and is often leveraged in run-time profiling and optimization systems [7, 69].

We use WinPAPI, the Windows variant of PAPI [21], to access the HPCs present in the processor. In our study we concentrate on the nine specific performance metrics listed in Table refperception:tab:hpcs. These counters are manually selected as a representative set of the HPCs available on the Pentium M. The choice of using only nine counters is due to a WinPAPI limitation. We collect counter values every 100 ms. WinPAPI automatically time multiplexes and scales the nine event counters.

2.2. User Study Setup

To explore the relationships between different microarchitectural parameters and user satisfaction, we conduct two sets of studies with 20 users. Our experiments are done

using an IBM Thinkpad T43p with a 2.13 GHz Pentium M-770 CPU and 1 GB memory running Microsoft Windows XP Professional SP2. The laptop is tethered to the power outlet during all experiments. Although eight different frequency levels can be set on the Pentium M-770 processor, only six can be used due to limitations in the SpeedStep technology. For both user studies, we experiment with three types of applications: a 3D Shockwave animation, a Java game, and high-quality video playback. The details of these applications follow:

- **Shockwave:** Watching a 3D Shockwave animation using the Microsoft Internet Explorer web browser. The user watches the animation and is encouraged to press the number keys to change the cameras viewpoint. The animation is stored locally. Shockwave options are configured so that rendering is done entirely in software on the CPU.
- **Java Game:** Playing a Java based First Person Shooter (FPS). The users have to move a tank and destroy different targets to complete a mission. The game is CPU-intensive.
- **Video:** Watching a DVD quality video using Windows Media Player. The video uses high bandwidth MPEG-4 encoding.

Since we target the CPU in this paper, we picked three applications with varying CPU requirements: the Shockwave animation is very CPU-intensive, the Video places a relatively low load on the CPU, and the Java game falls between these extremes.

Our user studies are double-blind, randomized, and intervention-based. We developed a user pool by advertising our studies within Northwestern University. While many of

the participants were CS, CE, or EE graduate students, our users included inexperienced computer users as well.

2.3. Correlation Between User Satisfaction and Hardware Counters

The primary objective of our first user study is to explore the correlation between HPCs and user satisfaction. The monitored hardware counters are listed in Table 2.1. In this first set of experiments, the users are asked to carry out the three application tasks as described in Section 2.2. During execution, we randomly change the frequency and ask the users to verbally rank their experience on a scale of 1 (discomfort) to 10 (very comfortable). Users typically provided a satisfaction rating within 5–10 seconds. These satisfaction levels are then recorded along with the HPC readings and analyzed as described in the next section. Then we compute the maximum, minimum, average, range, and the standard deviation of the counter values for up to 5 seconds within the given interval. The end result is a vector of 45 metrics for each satisfaction level reported by the user. Note that since we have performed the user studies with 20 users and three applications, we collected 360 user satisfaction levels.

We then find the correlation of the 45 metrics to the user satisfaction rating by using the formula:

$$(2.1) \quad r_{x,y} = \frac{N \sum xy - (\sum x)(\sum y)}{\sqrt{[N \sum x^2 - (\sum x)^2][N \sum y^2 - (\sum y)^2]}}$$

Pearsons Product Moment Correlation Coefficient (r) is commonly used to find correlation among two data series (x and y) and results in a value between -1 and 1. If the

correlation is negative, the series have negative relationship; if it's positive, the relationship is positive. The closer the coefficient is to either 1 or -1, the stronger the correlation between the variables. Thus, the magnitude of these correlations allows us to compare the relative value of each independent variable in predicting the dependent variable. The correlation factors for each of the 45 parameters and the user rating are presented in Section 2.7. In summary, we observe a strong correlation between the hardware metrics and user satisfaction rating: there are 21 parameters that correlate with the user satisfaction rating by a factor above 0.7 (all these 21 parameters have a factor ranging between 0.7 and 0.8) and there are 35 parameters with factors exceeding 0.5. On one hand, this result is intuitive; it is easy to believe that metrics representing processor performance relate to user satisfaction. On the other hand, observing the link between such a high-level quantity as measured user satisfaction and such low-level metrics as level 2 cache misses is intriguing.

We classify the metrics (and their correlations with user satisfaction) based on their statistical nature (mean, maximum, minimum, standard deviation, and range). The mean and standard deviation of the hardware counter values have the highest correlation with user satisfaction rating. A t-test analysis shows with over 85% confidence that mean and standard deviation both have higher r values when compared to the minimum, maximum, and range of the HPC values.

We analyze the correlations between the satisfaction results and user. Note that the r value cannot be used for this purpose, as the user numbers are not independent. Instead, we repeatedly fit neural networks to the data collected for each application, attempting to learn the overall mapping from HPCs to user satisfaction. As the inputs to the neural

network, we use the HPC statistics along with a user identification for each set of statistics. The output is the self-reported user satisfaction rating. In each fitting, we begin with a three-layer neural network model using 50 neurons in the hidden layer (neural networks are described in more detail in Section 2.4.2). After each model is trained, we perform a sensitivity analysis to find the effect of each input on the output. Sensitivity analysis consists of making changes at each of the inputs of the neural network and observing the corresponding effect on the output. The sensitivity to an input parameter is measured on a 0 to 1 scale, called the *relative importance factor*, with higher values indicating higher sensitivity. By performing sensitivity analysis, we can find the input parameters that are most important in determining an output parameter, i.e., user satisfaction. During this process, *we consistently find that the user number input has by far the highest relative importance factor*. Averaging across all of our application tasks, the relative importance factor of the user number is 0.56 (more than twice as high as the second factor). This strongly demonstrates that the user is the most important factor in determining the rating.

Finally, to understand the nature of the relationship between the HPCs and the user satisfaction, we analyze the trends for different functions for user satisfaction as provided by the user at each of the processor frequencies.

Figure 2.2 summarizes the trends observed among different users for our three applications. The first row shows the trend curves when we plot user satisfaction against the different frequencies (along x-axis). Most of the trends can be placed in four major categories:

- **Constant:** User satisfaction remains unchanged with frequency. As a result, it is not affected by frequency setting.

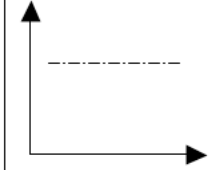
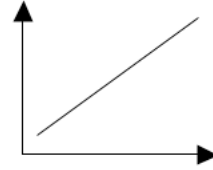
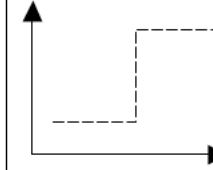
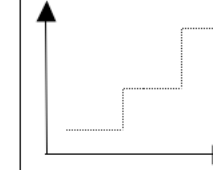
Applications	 Constant	 Linear	 Step	 Staircase	Other
Java Game	0	4	8	4	4
Shockwave	1	5	7	6	1
Video	18	0	0	0	2

Table 2.2. User trend categorization, the number of users in each category for different applications.

- **Linear:** User satisfaction increases linearly with processor frequency.
- **Step:** User satisfaction is the same for a few high frequencies but then plummets suddenly for the remaining lower ones.
- **Staircase:** User satisfaction takes on discrete values that monotonically increase with increasing frequency.

User satisfaction functions that do not match any of the above categories are labeled Other. Usually, this is due to user feedback which provides a non-monotonic function

These results reveal several important trends. First, user satisfaction is often non-linearly related to processor frequency. The majority of users provide functions that are categorized as Constant, Step, or Staircase. Note that although Constant is a linear function, it does not follow the regular assumption that an increase in a given metric results in an increase in user satisfaction. Second, user satisfaction is application-dependent. For example, for the Video application, almost all of the users report a Constant function. On the other hand, the trends for the Java game are distributed among various categories. Finally, user satisfaction is user-dependent. For example, in both the Java game, and the

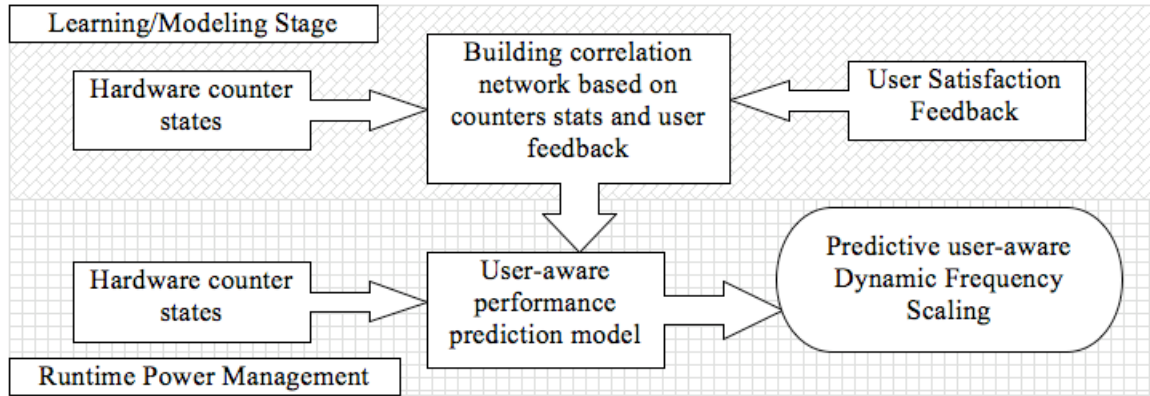


Figure 2.1. Framework of the predictive user-aware power management.

Shockwave animation, users specify utility functions that span multiple categories. This shows that different users have significantly different expectations for the system.

As we will discuss in the next section, these observations have an important effect on the modeling technique we use for learning and predicting user satisfaction. Overall, this motivational study indicates that

- Hardware counter have a strong correlation with user satisfaction;
- The individual user is the most important factor in determining user satisfaction;
- The relation between hardware performance and user satisfaction is often non-linear; and
- User satisfaction is both application dependent and user dependent.

Based on these observations, we design, implement, and evaluate a DVFS scheme that is based on individual user preferences.

2.4. Leveraging User Variation with Predictive Power Management

Based on the initial user study results presented in Section 2.3, we develop a power management scheme that sets the frequency of the processor based on estimates of user

satisfaction. This section presents this predictive user-aware power management scheme, called *Individualized Dynamic Frequency and Voltage Scaling (iDVFS)*. To implement iDVFS, we have built a system that is capable of predicting a users satisfaction based on interaction with the system. The framework can be divided into two main stages as depicted in Figure 2.1:

- **Learning Stage:** The system is initially trained based on reported user satisfaction levels and HPC statistics as described in Section 2.3. Machine learning models, specifically artificial neural networks, are trained offline to learn the function from HPC values to user satisfaction.
- **Runtime Power Management:** Before execution, the learned model is loaded by the system. During run time, the HPC values are sampled, entered into the predictive model, and then the predicted user satisfaction is used to dynamically set the processor frequency.

2.4.1. Learning Stage

In its learning stage, our algorithm builds a predictive model based on individual user preferences. The model estimates user satisfaction from the HPCs. In this stage, the user is asked to give feedback (user satisfaction level) while the processor is set to run at different frequency levels. The nature of this training stage is similar to the user study described in Section 2.2 and Section 2.3. Note that the user study and its survey are repeated for each application. While a user study runs, the nine performance counters are collected and the 45 statistical metrics computed from them are extracted. The

combination of these values and the user feedback are used to build the model that will later be used online.

2.4.2. Predictive Model Building

The learning stage helps us gather data that associates an individual user's satisfaction with different hardware performance counter readings and statistics. These instances are then used to build a predictive model that estimates the satisfaction of a particular user from the HPCs. We use neural networks to learn this model. We have also experimented with regression models and decision trees, but the neural networks provided the highest accuracy.

An *artificial neural network* (ANN) is an interconnected group of artificial neurons that uses a mathematical or computational model for information processing based on a connectionist approach to computation. An ANN maps a set of p input variables x_1, \dots, x_p to a set of q response variables y_1, \dots, y_q . It works by simulating a large number of interconnected simple analog processing units that resemble abstract versions of a neuron. Each processing unit (or neuron) computes a weighted sum of its input variables. The weighted sum is then passed through the sigmoid function to produce the unit's output. We use a three-layer ANN model with one input layer, one hidden layer, and one output layer. The well-known *backpropagation algorithm* is used to train the neural network from instance data. In the backpropagation algorithm, the weights between the neurons begin as random values. During the learning phase, training inputs are provided to the ANN and the associated output errors are used to adjust neuron weight functions to reduce error.

Our experiments represent an interesting case for machine learning. Typically, machine learning algorithms are extensively trained using very large data sets (e.g., thousands of labeled training inputs). We would like to use ANNs for their ability to learn complex non-linear functions, but do not have a very large data set. For each application-user pair, we only have six training inputs; one for each processor frequency. A training input consists of a set of HPC statistics and a user-provided satisfaction label. When we first began building ANN models with all 45 inputs (9 HPC counters with 5 statistics each), we noticed that our models were overly conservative, only predicting satisfaction ratings within a narrow band of values. We used two training enhancements to permit the construction of accurate ANN models. First, we simplified the ANN by limiting the number of inputs. Large ANNs require large amounts of training data to sufficiently learn the weights between neurons. To simplify the ANN, we used the two counters that had the highest correlation, specifically PAPI_BTAC_M-avg and PAPI_TOT_CYC-avg (as shown in Section 2.7). Second, we repeatedly created and trained multiple ANNs, each beginning with different random weights. After 30 seconds of repeated trainings, we used the most accurate ANN model. These two design decisions were important in allowing us to build accurate ANN models.

2.4.3. HPC-based Frequency Control Algorithm

*i*DVFS uses ANN models to determine the frequency level. The decision is governed by the following variables: f , the current CPU frequency; μ_{US} , the user satisfaction prediction for the last 500 ms of execution as predicted by the ANN model; ρ , the satisfaction tradeoff threshold; α_f , a per-frequency threshold for limiting the decrease of frequency

from the current f ; M , the maximum user comfort level; and T_i , the time period for re-initialization.

i DVFS employs a greedy approach to determine the of M , i DVFS predicts that the frequency is in a satisfactory state. If μ_{US-1} , the previously predicted user comfort, is also of M , the system determines that it may be good to decrease the processor frequency; if not, then the system of M , then the system determines that the current performance is not satisfactory and increases the operating frequency. i DVFS uses the α_f thresholds as a hysteresis mechanism to eliminate the ping-pong effect between two states. If the processor rapidly switches between two states N times in a short time interval, the appropriate α_f threshold is decreased to make it harder to decrease to the lower frequency level. This feature of the algorithm ensures that i DVFS can adjust to a set of operating conditions very different from those present at initialization but at a rate that is maximally bounded by T_i . The constant parameters ($\rho = .15$, $N = 3$, $T_i = 20$ seconds) were set based on the experience of the authors using the system. α_f thresholds are initialized to 1 for each of the frequency level and is decremented by 0.1 at each frequency boost

Ideally, we would like to empirically evaluate the sensitivity of i DVFS performance to the selected parameters. However, it is important to note that any such study would require having real users in the loop, and thus would be slow. Testing four values of four parameters on 20 users would require 256 days (based on 20 users/day and 25 minutes/user). For this reason, we decided to choose the parameters based on qualitative evaluation by the authors and then “close the loop” by evaluating the whole system with the choices.

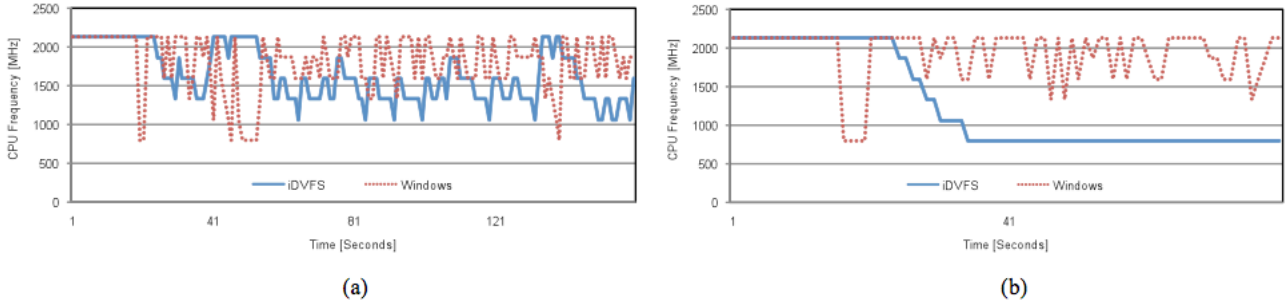


Figure 2.2. Figure 2. Frequency traces using *iDVFS* and Windows power management schemes for (a) Java Game and (b) Video.

Figure 2.2 illustrates the performance of the *iDVFS* algorithm for two of the three applications in our study. Each graph shows, as a function of time, the CPU frequency for a randomly-selected user when playing the Java Game and watching the Video. First, note that the frequency transitions in the two example traces differ greatly from the decisions that Windows XP DVFS makes. The reason is that Windows XP DVFS alters frequency based upon CPU utilization while *iDVFS* alters frequency based upon predicted user satisfaction. *iDVFS* reduces the frequency significantly in the Video application. In this case, the user has indicated high satisfaction with all levels of performance. As shown in Figure 2.2, the Video has the least variation in user satisfaction values at lower frequencies. As a result the *iDVFS* algorithm can reduce CPU frequency without affecting user satisfaction. In both cases, the frequency level follows the satisfaction levels reported by the user and minimizes power consumption with little impact on satisfaction. These traces show that *iDVFS* can successfully adjust the clock frequency throttle according to the user satisfaction derived from the HPCs. For a highly compute-intensive application (such as the Java Game), the reduction in the frequency is minimal because any change in frequency causes a significant reduction in userperceived performance. For other

applications (such as the Video), frequency can be drastically reduced without affecting user satisfaction.

2.4.4. Implementation, Integration, and Limitations

Currently, we have not integrated *iDVFS* with the operating system (OS). Instead, we have implemented client software that runs as a Windows toolbar task, and manually activate *iDVFS* for our user studies. The client is implemented in a manner that is similar to profile-directed optimization. An initial calibration stage is used for building a model that is used to predict user satisfaction during run time. The current implementation requires direct user feedback in a calibration stage for each user and each application. While this may be cumbersome, there are two points we would like to make. First, we believe that the current system is practical for some users (e.g., heavy gamers will not mind a few minutes of calibration). Second, we argue that explicit user feedback is a viable option. Future work in limiting the feedback and learning effectively from explicit/implicit mechanisms will allow such schemes to be deployed widely.

iDVFS has a few limitations that will be eliminated once it is integrated into the OS. First, we provide the client software with per-user, per-application neural network models tailored to the application we are about to invoke. Second, *iDVFS* is currently only intended for interactive applications. The OS has knowledge of users, as well as active applications, and could automatically load the appropriate prediction models for interactive applications during context switches.

WinPAPI only supports system-wide HPC sampling; this includes other programs, background processes, and kernel execution. For our work, we run a single workload on the

machine at a time; hence HPC samples correlate to the workload directly. Ideally, the HPC interface would include thread-specific information as well as distinguish between user level and kernel level applications. Other HPC interfaces (i.e., perfmon2 for Linux [53]) also include this support.

The performance of *iDVFS* is largely dependent upon good user input. While this may be a limitation for a current user and application, the user is free to provide new ratings and recalibrate *iDVFS* if the resulting control mechanism causes dissatisfaction.

2.5. Experimental Results

In this section, we evaluate the predictive user-aware power management scheme with a user study, as described in Section 2.4. We compare *iDVFS* with the native Windows XP DVFS scheme and report reductions in CPU dynamic power, as well as changes in measured user satisfaction. This is followed by a trade-off analysis between user satisfaction and system power reduction. We report the effect of *iDVFS* on the power consumption and user satisfaction.

We compare *iDVFS* to Windows Adaptive DVFS, which determines the frequency largely based on CPU usage level. A burst of computation due to, for example, a mouse or keyboard event brings utilization quickly up to 100% and drives frequency, voltage, power consumption, and temperature up along with it. CPU-intensive applications cause an almost instant increase in operating frequency and voltage regardless of whether this change will impact user satisfaction. Windows XP DVFS uses six of the frequency states in the Enhanced Intel Speedstep technology, as mentioned in Section 3. Performance

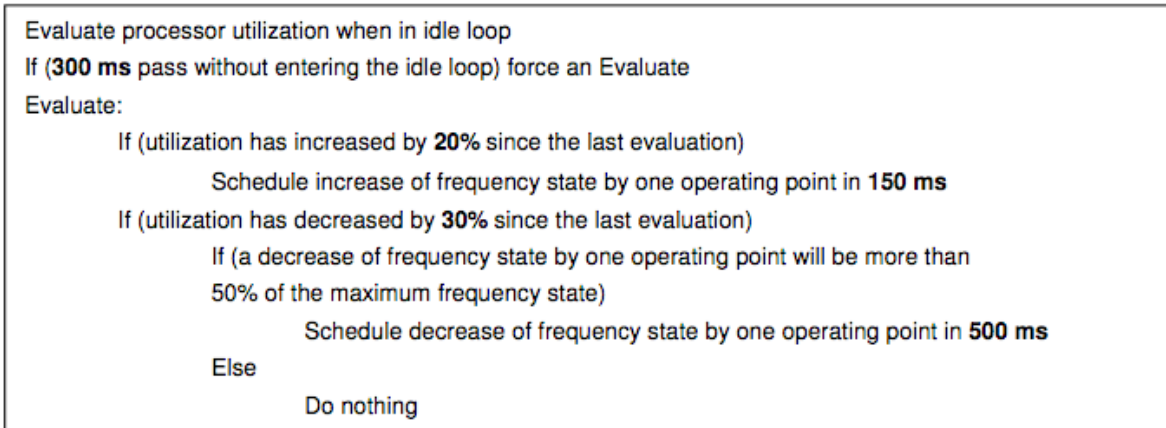


Figure 2.3. Windows DVFS algorithm.

requirements are determined using heuristics based on metrics “such as processor utilization, current battery level, use of processor idle states, and inrush current events” [76]. In the Windows native adaptive DVFS scheme, decisions are made according to the algorithm described in Figure 2.3. We note that this is our best interpretation of the DVFS algorithm described in [76].

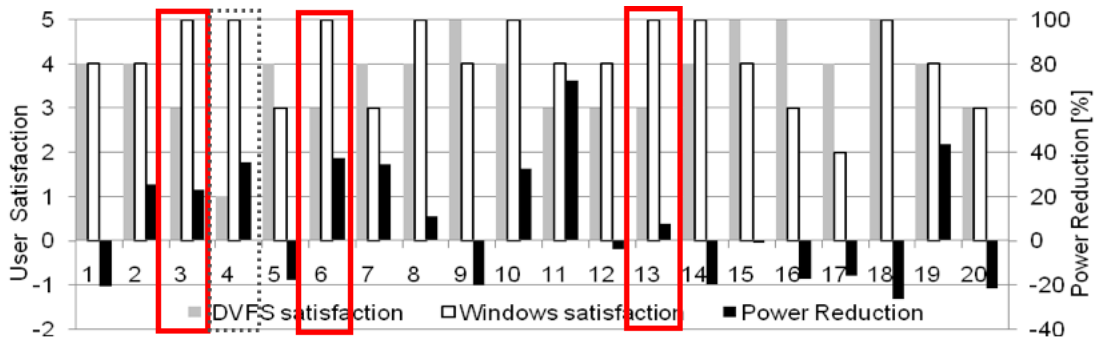
2.5.1. Analysis of User Satisfaction and Power Measurements

To analyze the effect of *i*DVFS on system power consumption, we perform a second set of user studies in which the users are asked to carry out the tasks described in Section 2.2. This time, the durations of the applications are increased: the Java Game is executed for 2.5 minutes; Shockwave and Video are executed for 1.5 minutes each. The user is asked to execute the application twice, once for Windows XP DVFS and once for *i*DVFS, which loads the individual ANN model for the user/application before the start of the execution. Once the execution completes, the users are asked to rate their satisfaction with each of the systems on a scale of 1 (very dissatisfied) to 5 (very satisfied).

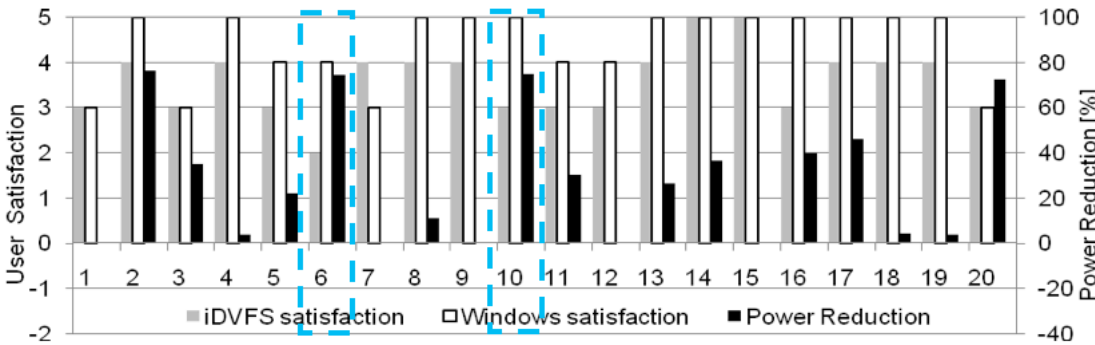
During these experiments, we log the frequency over time. We use these frequency logs to derive CPU power savings for *i*DVFS compared to the default Windows XP DVFS strategy. We have also measured the online power consumption of the entire system, and provide a detailed discussion and analysis of trade-offs between power consumption and user satisfaction.

2.5.1.1. Dynamic Power Consumption and User Satisfaction. The dynamic power consumption of a processor is directly related to frequency and supply voltage and can be expressed using the formula $P = V^2CF$, which states that power is equal to the product of voltage squared, capacitance, and frequency. By using the frequency traces and the nominal voltage levels on our target processor [45], we calculated the relative dynamic power consumption of the processor. Figure 2.4 presents the CPU dynamic power reduction achieved by the *i*DVFS algorithm compared to the Windows XP DVFS algorithm for the individual users for each application. It also presents their reported satisfaction levels. To understand the figure, consider a group of three bars for a particular user. The first two bars represent the satisfaction levels for the users for the *i*DVFS (gray) and Windows (white) schemes, respectively. The third bar (black) shows the power saved by *i*DVFS for that application compared to the Windows XP DVFS scheme (for which the scale is on the right of the figure).

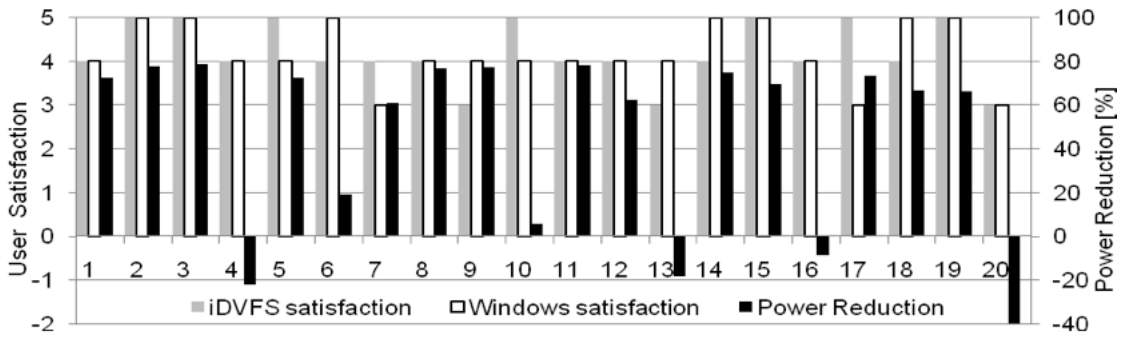
On average, our scheme reduces the power consumption by 8.0% (Java Game), 27.9% (Shockwave), and 45.4% (Video) compared to the Windows XP DVFS scheme. A one-sample t-test of the *i*DVFS power savings shows that for Shockwave and Video, *i*DVFS decreases dynamic power with over 95% confidence. For the Java game, there are no statistically-significant power savings. Correspondingly, the average user satisfaction level is reduced



(a) Java game.



(b) Shockwave animation.



(c) Video.

Figure 2.4. Reported user satisfaction for the (a) Web Browsing, (b) Game, and (c) Video applications.

by 8.5% (Java Game), 17.0% (Shockwave), and remains the same for Video. A two-sample paired t-test comparing the user satisfaction ratings from *i*DVFS and Windows

XP DVFS indicates that for Java and Video, there is no statistical difference in user satisfaction when using *iDVFS*. For Shockwave, we reduce user satisfaction with over 95% confidence

The combined results show that for Java, *iDVFS* is no different than Windows XP DVFS, for Shockwave, *iDVFS* trades off a decrease in user satisfaction for a decrease in power consumption, and for the Video, *iDVFS* significantly decreases power consumption while maintaining user satisfaction.

An analysis of the results quickly reveals that the average satisfaction levels are strongly influenced by a few exceptional cases. We have analyzed the cases where there is a difference of more than 1 step between the user ratings. Among these, we found six cases that require special attention. For the Java Game, the training inputs of Users 3, 6, and 13 (solid rectangles in Figure 4) significantly mismatched the performance levels of the processor. Specifically, these users have given their highest ratings to one of the lowest frequency levels. As a result, *iDVFS* performs as the user asks and reduces the frequency, causing dissatisfaction to the user. The cause of dissatisfaction for User 4 (dotted rectangle in Figure 2.4) was different. The ANN for that user did not match the training ratings and thus the user was dissatisfied. Similarly, for the Shockwave application, Users 6 and 10 (dashed rectangle in Figure 2.4) provided a roughly constant user satisfaction across the various frequencies. During the user study, however, these Shockwave users highlighted their dissatisfaction when they were able to compare the performance of *iDVFS* to the Windows scheme, which keeps the processor at the highest frequency at all times

It is important to note that such exceptional cases are rare; only 10% of the cases (6 out of 60) fall into this category. Such exceptional cases can be easily captured during a

learning phase and eliminated by forcing the user to retake the survey and re-train the model, i.e., training can be repeated until successful. In addition, any dissatisfied user can retrain until a satisfactory performance level is reached. However, our results reveal that such cases will be rare ¹

User 16s results are likely to be caused by noise and provide a good example of the intricacies of dealing with real users. This user rated *iDVFS* two steps lower than the Windows scheme for Shockwave. At the same time, he/she rated *iDVFS* two grades higher for the Java Game application even though *iDVFS* used a lower frequency throughout execution.

Overall, these initial results provide strong evidence that a highly-effective individualized power management system can be developed. Specifically, the results from our user study reveal that

- There exist applications (e.g., Video), for which providing customized performance can result in significant power savings without impacting user satisfaction;
- There exist applications (e.g., Shockwave), for which the users can trade off satisfaction level with power savings. In fact, in the next section, we provide an analysis of such trade-offs; and
- There exist applications (e.g., Java Game), for which traditional metrics in determining the satisfaction is good and *iDVFS* will provide the same performance level and user satisfaction.

¹We also analyzed the performance of *iDVFS* without considering these extreme cases. Overall, *iDVFS* reduces power consumption by 5.2% (Java Game), 24.0% (Shockwave), and 45.4% (Video). User satisfaction levels were increased by 4.8% (Java Game), reduced by 13.9% (Shockwave), and remained identical for Video (where there are no exceptional cases).

2.5.1.2. Total System Power and Energy-Satisfaction Trade-Off. In the previous section, we have presented experimental results indicating the user satisfaction and the power consumption for three applications. For two applications (Video and the Java Game), we concluded that the *i*DVFS users are at least as satisfied as Windows XP DVFS users. However, for the Shockwave application, we observed that although the power consumption is reduced, this is achieved at the cost of a statistically significant reduction in average user satisfaction. Therefore, a designer needs to be able to evaluate the success of the overall system. To analyze this trade-off, we developed a new metric called the energysatisfaction product (ESP) that works in a similar fashion to popular metrics such as energy-delay product. Specifically, for any system, the ESP per user/application can be found by multiplying the energy consumption with the reported satisfaction level of the user.

Clearly, to make a fair comparison using the ESP metric, we have to collect the total system energy consumption during the run of the application. To extract these values, we replay the traces from the user studies of the previous section. The laptop is connected to a National Instruments 6034E data acquisition board attached to the PCI bus of a host workstation running Windows (and the target applications), which permits us to measure the power consumption of the entire laptop (including other power consuming components such as memory, screen, hard disk, etc.). The sampling rate is set to 10Hz. Figure 2.5 illustrates the experimental setup used to measure the system power.

Once the system energy measurements are collected (for both Windows XP DVFS and *i*DVFS), we find the ESP for each user by multiplying their reported satisfaction levels and the total system energy consumption. The results of this analysis are presented



Figure 2.5. System power measurement setup.

in Figure 2.6. In this figure, we present the reduction in system energy consumption, increase in user satisfaction, and change in ESP for each user. Hence, the higher numbers correspond to improvement in each metric, whereas negative numbers mean that the Windows XP DVFS scheme performed better. Although the ESP improvement varies from user to user, we see that *i*DVFS improves the ESP product by 2.7%, averaged over all users. As a result, we can conclude that Windows XP DVFS and *i*DVFS provide comparable ESP levels for this particular application. In other words, the reduction in user satisfaction is offset at a significant benefit in terms of power savings.

2.6. Summary

Through extensive user studies, we have demonstrated that there is a strong, albeit usually nonlinear, link between low-level microarchitectural performance metrics, as measured by hardware performance counters (i.e., “close to the metal” numbers), and user satisfaction (i.e., “close to the flesh” numbers) for interactive applications. More importantly, we show that the link is highly user-dependent. This variation in user satisfaction

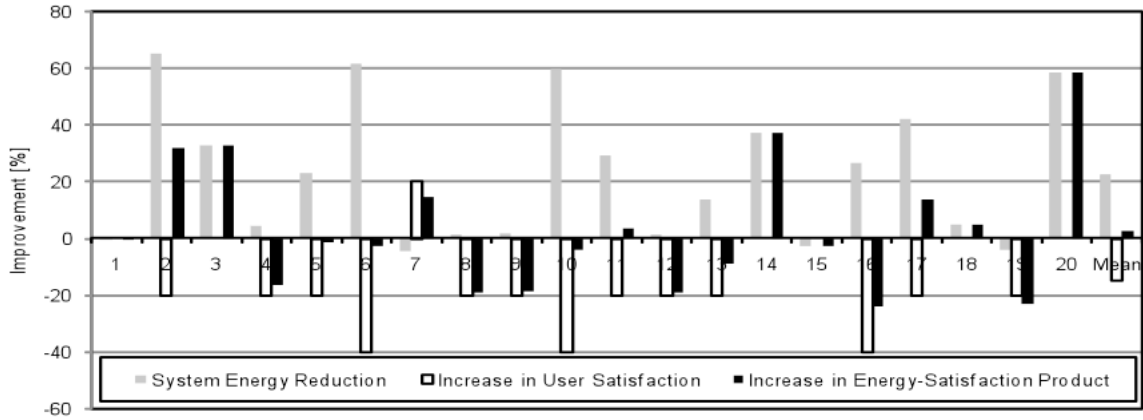


Figure 2.6. Figure 6. Improvement in energy consumption, user satisfaction, and energy-satisfaction product for the Shockwave application.

indicates potential for optimization. Using neural networks, we learn per-user perapplication functions (which might be called “metal to flesh” functions) that map from the hardware performance counters to individual user satisfaction levels. This result in a computer system that can uses small amounts of explicit user feedback, and then implicitly learns from the feedback to make online predictions of user satisfaction. We demonstrate the utility of this implicit feedback by employing it in a user-aware DVFS algorithm. Experimental results, and analysis of user studies, show that there are interactive applications for which knowledge of user satisfaction permits power consumption savings. Others present an interesting trade-off between user satisfaction and power savings. Overall, our system reduces the power consumption of Windows XP DVFS by over 25%, while only affecting user satisfaction in one application.

2.7. Appendix: Correlating HPC Metrics and User Satisfaction

Table 2.3 presents the correlation between 45 metrics based on hardware counter readings. Please see Section 4 for details of the calculation of these correlation factors.

Performance Metrics	Correlation	Performance Metrics	Correlation	Performance Metrics	Correlation
PAPLBTAC_M-avg	0.771	PAPLRES_STL-max	0.738	PAPLTOT_INS-range	0.625
PAPLL1_ICA-avg	0.770	PAPLBTAC_M-max	0.733	PAPLTOT_INS-min	0.603
PAPLL1_ICA-stdev	0.770	PAPLTOT_INS-max	0.729	PAPLL1_DCA-min	0.528
PAPLBTAC_M-stdev	0.770	PAPLL2_TCM-avg	0.722	PAPLL2_TCM-max	0.525
PAPLL1_DCA-stdev	0.768	PAPLL1_DCA-range	0.721	PAPLBR_MSP-min	0.503
PAPLTOT_INS-avg	0.768	PAPLL2_TCM-stdev	0.709	PAPLL2_TCM-range	0.497
PAPLTOT_CYC-avg	0.767	PAPLRES_STL-min	0.694	PAPLL2_TCM-min	0.495
PAPLL1_DCA-max	0.767	PAPLTOT_CYC-min	0.689	PAPLBR_MSP-max	0.379
PAPLTOT_CYC-stdev	0.767	PAPLRES_STL-range	0.684	PAPLBR_MSP-range	0.360
PAPLTOT_INS-stdev	0.766	PAPLL1_ICA-min	0.682	PAPLBTAC_M-min	0.289
PAPLL1_DCA-avg	0.766	PAPLL1_ICA-range	0.675	PAPLHW_INT-max	0.131
PAPLRES_STL-avg	0.761	PAPLBR_MSP-avg	0.662	PAPLHW_INT-range	0.119
PAPLRES_STL-stdev	0.761	PAPLBTAC_M-range	0.653	PAPLHW_INT-min	0.112
PAPLTOT_CYC-max	0.756	PAPLTOT_CYC-range	0.644	PAPLHW_INT-stdev	0.094
PAPLL1_ICA-max	0.749	PAPLBR_MSP-stdev	0.638	PAPLHW_INT-avg	0.048

Table 2.3. Correlation of hardware counters and user satisfaction.

CHAPTER 3

User Physiological Traits: Implicitly Learning User Satisfaction

Knowing that user variation exists is not enough. We need a method for understanding user satisfaction during computer use. Although *iDVFS* (described in Section 2.4) uses a model from hardware performance counters to user satisfaction as a proxy for predicting user satisfaction, we still need to explicitly ask the end user for their satisfaction at some point. This explicit input may be annoying to the end user, and begs the question: is it possible to *implicitly* understand user satisfaction? If we could understand the end user without explicit feedback, we could drive user-aware optimizations, such as *iDVFS*, to tune performance to an individual user without any explicit feedback.

Developing an implicit feedback mechanism for modern computers is difficult. We assert that the design of modern architectures makes it difficult (if not impossible) to implicitly infer and reason about the end user. One only needs to observe the current computer usage model to understand this claim. First, the user directs the computer explicitly via input devices (e.g., keyboard or mouse). According to user direction, the computer executes instructions to manipulate machine state. Afterwards, the user obtains information via output devices (e.g., display or speakers). Note that during this human-computer interaction, *there is a considerable asymmetry between the information available to the user and information available to the computer*. Although the user can direct the computer to change/view the system state at any time, the computer executes with little any information about the user state.

In this chapter, we make a case for balancing this human-computer information asymmetry by augmenting future architectures with new input devices that provide information on user state. Enabling a computer to sense and perceive user state has a number of benefits. First, understanding user state will enable user-aware optimizations by providing implicit user feedback. Tailoring execution to the individual users “taste” will result in better efficiency and significant benefits in power savings or increased lifetime reliability. In addition, decisions about resource assignment (i.e., deciding on the level of parallelism of an application running on a chip multiprocessor) can be made more effectively. Most importantly, computer behavior will be personalized based upon individual expectations to improve user satisfaction.

We propose, and evaluate, the use of biometric input devices that provide information on human state by observing physiological traits. Using physiological readings is an intuitive first step in understanding the user; our experiments suggest that a change in user state results in a number of measurable physiological responses. We use an eye tracker to measure pupil dilation and eye movement, a galvanic skin response (GSR) sensor to measure skin resistance/conductance, and force sensors to measure behavior. We begin with two user studies to motivate the use of these additional input devices. In the first, we drastically drop the CPU frequency at a set point while a game is being played. In the second, we randomly vary the CPU frequency across multiple settings during game play. We show that the CPU frequency has a significant impact on the physiological traits of the users. We also show that the changes in the physiological traits correlate with the satisfaction levels reported by the participants.

Based upon these observations, we then construct a *Physiological Traits-based Power-management* (PTP) system to demonstrate an application of these biometric input devices. PTP may augment any existing dynamic voltage and frequency scaling (DVFS) scheme to make user-aware decisions. In its current implementation, PTP adjusts the maximum frequency by incorporating human physiological readings. DVFS is a common power saving technique available on modern microprocessors that scales the frequency (and voltage) of a microprocessor to reduce power consumption. By adding PTP to a typical CPU-utilization-based DVFS scheme, we significantly decrease power consumption with little to no impact on user satisfaction.

It is intuitive to imagine that the computer performance will impact the physiological responses of users. There have been studies showing the relationships between physiological sensor readings and reported user emotions in response to interaction with computer programs [75, 52]. However, to the best of our knowledge, this is the first study in measuring the impact of computer performance on human physiological traits. Specifically, we make the following contributions:

- We make a case for using biometric input devices (such as eye trackers, galvanic skin response sensors, and force sensors) in making architecture-level decisions;
- We show through two user studies that our selected biometric input devices are able to detect changes in human physiological traits as the performance is altered during the run of an application; and
- We demonstrate a user-aware system for augmenting DVFS and evaluate the system with another user study.

The rest of the chapter is organized as follows. Section 3.1 discusses the biometric sensors. Section 3.2 presents the setup of the user studies. Section 3.3 describes the first two user studies correlating sensor readings to user satisfaction. Section 3.4 discusses our prototype DVFS system for leveraging biometric input devices. Section 3.5 presents our results, We conclude with Section 3.6.

3.1. Empathic Input Devices

To support user-aware computer architectures, computers will require a means to understand user satisfaction. Although it is possible to explicitly ask the user for information, this may be annoying. The ability to implicitly determine the degree of user satisfaction would be ideal. Unfortunately, current architectures are not equipped to implicitly estimate user satisfaction. This is due to a fundamental limitation of current input devices. Traditional input devices mainly exist to allow the user to explicitly control the machine state. However, they provide little information about physiological state. Without any information about user state, it is obvious that a computer cannot reason about user satisfaction. To help bridge this gap, we make a case for the addition of biometric sensors in future architectures. In this work, we explore three biometric sensors: eye trackers, galvanic skin response sensors, and force sensors. These sensors are described in the following sections.

3.1.1. Eye Tracker

Eye behavior reveals a lot of information about users state. We are particularly interested in pupil dilation and pupil movement. Pupil dilation, or changes in the pupil radius over

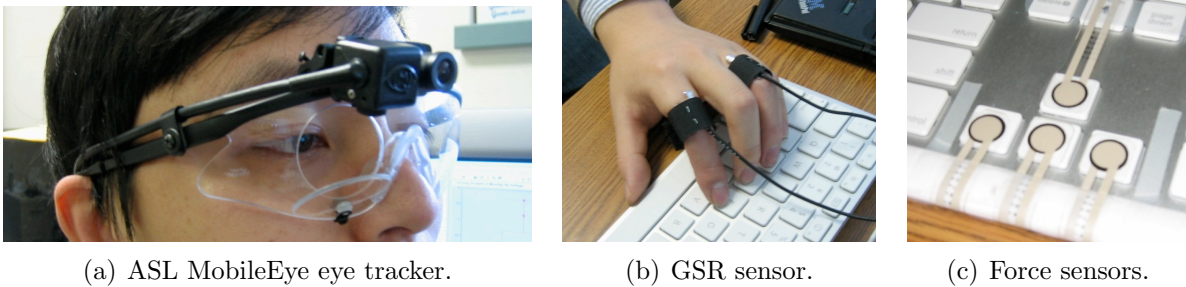


Figure 3.1. The biometric sensors used in our experiments: (a) an eye tracker, (b) a custom-made galvanic skin response sensor, and (c) force sensors attached to the arrow keys on the keyboard.

time, has been shown to correlate to many external and internal human factors. Studies show pupil dilation to be related to mental workload [56], perceptual changes [36], and positive/negative affect or emotion processing [82]. Pupil movement provides another source of information. Even when viewing a still image, humans do not keep their eyes steady. Instead, the eye constantly looks around finding interesting parts of each scene to create a larger mental map of the whole scene. Changes in the behavior of eye movement may also indicate higher level changes in the scenery, or human interests/state. For example, saccades (fast simultaneous movement of both pupils) have been linked to boundaries of event perception [98].

We use the ASL MobileEye eye tracker, shown in Figure 3.1(a), for collecting eye-related information. The eye tracker uses video-based combined pupil/corneal reflection to track the focus of the users right eye. A video feed is analyzed to extract the pupil location and pupil radius. The data gathered is in pixels relative to the video feed, and is sampled 30 times per second. Pupil dilation is measured by using the pupil radius samples from the eye tracker. Pupil movement is measured using the Euclidean distance between consecutive samples of the pupil X-Y coordinates.

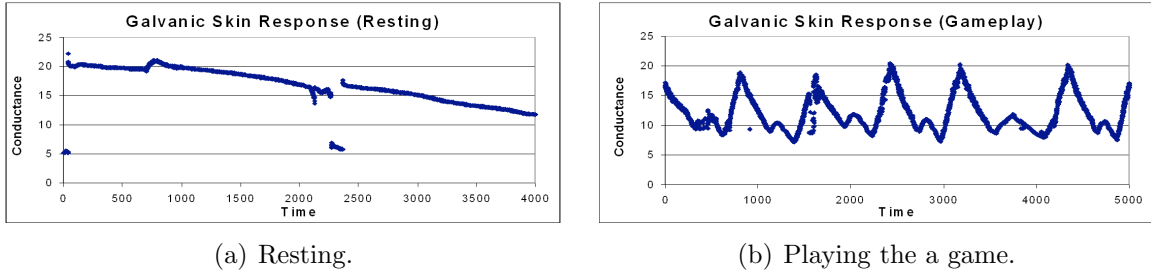


Figure 3.2. GSR traces of a user that capture (a) the long-term change in the GSR while a user is resting, and (b) the short-term effects when playing the Need for Speed game. The existence of the long-term effect motivates the use of the delta GSR metric for measuring user arousal.

3.1.2. Galvanic Skin Response

Galvanic skin response (GSR) [18] measures the skins ability to conduct electricity. GSR is strongly, but not completely, correlated to the conductance of sweat in sweat glands in skin [107]. GSR acts as an indicator of the autonomic nervous system reflecting both sympathetic (e.g., fight-or-flight response) as well as parasympathetic (e.g., rest or relaxation) response. In general, a low conductance is a sign of relaxation and high conductance is a sign of mental, emotional, and/or physical arousal. However, different emotions may produce discriminable waveforms [10, 105].

We use a custom-made GSR sensor which is shown in Figure 3.1(b). The GSR sensor consists of two probes attached to velcro strips that are wrapped around the users fingers during experiments. The two probes are wired in a voltage divider circuit for measuring the voltage (and therefore the resistance and/or conductance) across the skin. GSR readings show long-term and short-term effects. For example, two sample GSR traces for one of the authors are shown in Figure 3.2; Figure 3.2(a) shows the GSR when resting and Figure 3.2(b) shows the GSR when playing the Need for Speed computer game. At

rest, the GSR does not stay constant. Rather, it slowly decreases over a period of 5–10 minutes and then slowly levels out. When excited during game play, the GSR exhibits a much more varied response. To measure short-term changes in user arousal, and filter out the long-term trends, we employ a metric that we call *delta GSR*, which resembles the metric “hash GSR” [10]. Delta GSR is computed by taking the difference between consecutive samples and filtering out the negative values. When summed over a period of time, the delta GSR serves as a metric for the total user arousal for the time period. We sample at 30 Hz and use a period of one second.

3.1.3. Force Sensors

We also use force sensors (shown in Figure 3.1(c)) to collect behavioral information about the user. Studies in keystroke dynamics have shown that keystroke patterns for a given user are correlated with various emotional states [106]. However, the force of each key press might hold additional information not captured by timing alone. For example, users may press the keys harder to express annoyance, or during times of intense involvement in game play. Also, for some applications, the range of keys involved is quite limited, and force may provide more information than keystroke patterns. In this work, we study the correlation between keystroke force and user satisfaction.

We use force-sensitive resistors to instrument each of the four arrow keys, as shown in Figure 3.1(c). The force sensors are measured using a voltage divider circuit. The maximum pressure value among all measured keys yields a single metric for comparison, which we will refer to as *MaxArrow*. The sampling rate is 30 Hz.

3.1.4. Sensor Metrics

We measure four readings from the biometric input devices: pupil dilation, pupil movement, delta GSR, and arrow-key force. As we gather these readings, we summarize them using various statistics. For each reading, we consider the maximum, arithmetic mean, and the variance of the readings every second. We define the term sensor metric to be a specific combination of a statistic and a biometric reading. We format sensor metrics as follows: `<statistic>_<sensor>`. For example, the arithmetic mean of the pupil movement is denoted by `Mean_PupilMovement`.

3.1.5. Sensor Extensibility, Applicability, and Cost

The intrusiveness of sensors is a major consideration for using them as biometric input devices. Ideally, biometric input devices will (1) not impede the use of the computer in any way, (2) require little effort by the user, and (3) not incur significant financial cost. We select our sensors based on these principles. Consumer “remote eye tracking” products are available which detect eye focus and pupil radius without a head-mounted system. Further research into this area is likely to lower the cost of these systems [17]. Modern laptops contain built-in cameras and image recognition software exists for detecting pupils [79]. The electrical components required to measure GSR are inexpensive. While the velcro-strip contacts may be considered too cumbersome, these contacts have also successfully been integrated into a computer mouse in a way that requires no explicit action by the user [112]. Integrating force sensors into a computer keyboard would do little change to the existing structure and piezoresistive force sensors are inexpensive; the force sensors used for this work are currently available for under \$15 per sensor[104].

3.2. User Study Setup

Our experiments are done using an IBM Thinkpad T61 with a 2.2 GHz Intel Core 2 Duo T7500 processor and 2 GB DDR2 SDRAM running Microsoft Windows XP. The laptop is tethered to power for experiments. The processor supports seven frequency levels using Intel Enhanced SpeedStep Technology (2.2 GHz, 1.6 GHz, 1.2 GHz, 800 MHz, 600 MHz, 400 MHz, and 200 MHz). In our experiments, we use the top five frequencies ranging from 2.2 GHz to 600 MHz.

Data from the GSR and force sensors is collected using a National Instruments 603E data acquisition card connected to the PCI bus of a separate workstation. The workstation then sends the sensor information through a TCP socket to the laptop over a private LAN connection.

In our user studies, we use the following applications:

- **Need for Speed Pro Street [3]**: A 3D driving game against the computer. The game is very CPU-intensive.
- **Tetris Arena [2]**: A 3-D version of the classic puzzle game. The game consumes 100% of the CPU. However it exhibits little performance degradation as the frequency is decreased.
- **Microsoft Word 2000 Version 9.0 [1]**: The user is given a document to reproduce in Microsoft Word. In general, Microsoft Word is not CPU intensive. However, we include some high-quality images into the document. Moving the images occasionally causes short bursts of high CPU utilization.

We developed a user pool by advertising our studies within Northwestern University. The participants come from a variety of backgrounds and include males and females, engineers and non-engineers, as well as inexperienced computer users.

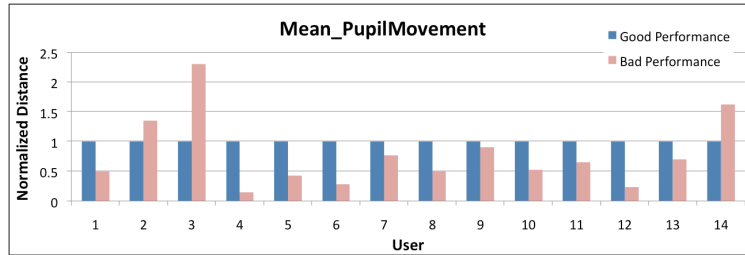
3.3. Correlating Human Physiological Traits with User Satisfaction

The ultimate goal of this paper is show how human physiological traits can be used as an implicit measure for inferring user satisfaction. In this section, we present two user studies exploring the link between human physiological readings and user satisfaction.

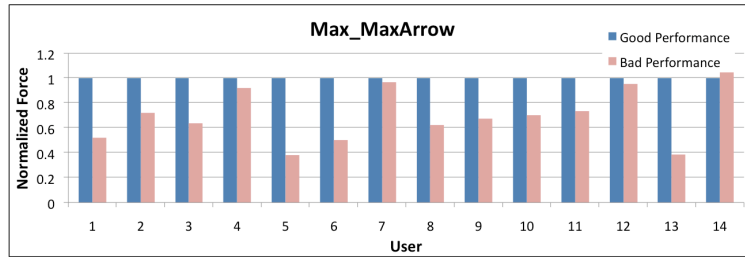
3.3.1. Motivating the Use of Physiological Sensors

The first user study explores whether there *are* changes in human physiological traits when the performance of the processor is changed. One of our major concerns was that the measurement noise during game play may mask any changes in physiological traits. It is not difficult to imagine possible sources of noise. For example, in a driving game, a difficult section of tight turns may produce different measurements than another section with a long straightaway. Due to this concern, we first conduct a controlled initial user study with 14 users. During the study, we ask the users to play the Need for Speed game twice. Each time, at a predetermined position on the racetrack, we either maintain the highest frequency, or drop the frequency to 600 MHz for 20 seconds. At 600 MHz, the game greatly slows down. During the 20 seconds, we measure statistics from each of the physiological sensors.

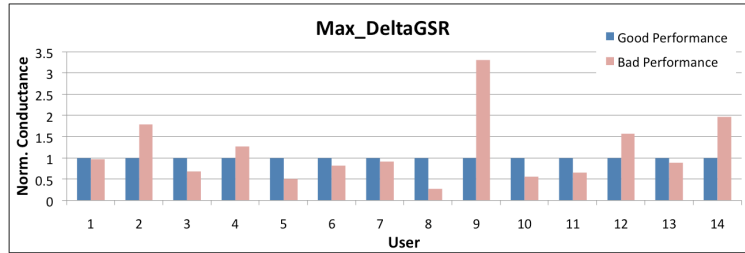
Figure 3.3 shows the data from three of the sensor metrics that display significant changes in the initial user study. Mean eye movement (shown in Figure 3.3(a)) decreases



(a) Mean eye movement



(b) Maximum force on the arrow keys.



(c) Maximum delta GSR.

Figure 3.3. (a) Mean pupil movement, (b) maximum arrow force, and (c) maximum delta GSR for the same 20 seconds of game play at a good performance level, and at a bad performance level. Mean pupil movement and maximum arrow force significantly decrease. Maximum delta GSR has more variation across users indicating different responses to a drop in performance.

for the large majority of the users. The maximum force on the arrow keys (shown in Figure 3.3(b)) also registers a noticeable decrease for most users. The maximum delta GSR (shown in Figure 3.3(c)) shows a relative change for many of the users. However, it increases for some users and decreases for others. The difference in users may be

attributed to varying emotional reactions to a slow system: some users become annoyed and more aroused, while others become bored and less involved. Nevertheless, the results indicate that both arousal-based sensors (e.g., DeltaGSR) and behavioral sensors (e.g., MaxArrow) do indeed change significantly as application performance is decreased.

3.3.2. Physiological Sensors and User Satisfaction

With the knowledge that the sensor metrics do indeed change with performance, we conduct a second study to explore (1) the effect of random game phases and (2) the correlation between physiological readings at different performance levels and user satisfaction. The users play the Need for Speed game. This time, the processor speed is changed to a random frequency at a random point in the game. The change in performance lasts for 30 seconds. We randomly visit each frequency level twice; the first time we collect sensor metric readings, and the second time we verbally ask the user for a satisfaction rating. Users report their satisfaction as follows: 5 (Very Satisfied), 4 (Satisfied), 3 (Indifferent), 2 (Unsatisfied), and 1 (Very Unsatisfied).

A good sensor metric will report as different when the user satisfaction changes and as similar when user satisfaction remains the same. To distinguish between sensor metrics at different frequencies, we employ a *t-test-based similarity metric*. As the physiological sensors are noisy by nature, we use multiple samples and statistical methods. Both the data acquisition card (collecting GSR and force information) and the eye tracker sample at 30 Hz. Each second, we compute the sensor metrics based on 30 samples. After discarding the first and last five seconds of each 30 seconds interval, we have 20 calculated values

Sensor Data	Success Rate	False Positive	False Negative
Max_PupilRadius	70.2%	14.3%	15.5%
Max_MaxArrow	69.0%	13.1%	17.9%
Mean_MaxArrow	69.0%	13.1%	17.9%
Mean_PupilRadius	67.9%	11.9%	20.2%
Mean_PupilMovement	57.1%	13.1%	29.8%
Max_DeltaGSR	58.3%	9.5%	32.1%

Table 3.1. Outcomes of manually comparing t-test results and the user satisfaction ratings. Success means that the t-test outcome matches the user rating. False negatives occur when the t-test falsely predicts a difference and false positives occur when the t-test falsely predicts similarity with the highest frequency.

per sensor metric. We then use a t-test, with a 90% confidence interval, as our metric for measuring the similarity between sets of values from different frequencies.

We now evaluate the behavior of our sensor metrics across multiple frequencies. For every sensor metric, we use the t-test-based similarity metric to compare each frequency with the highest frequency. The assumption is that if the user is annoyed, the t-test should indicate that the two sets are different; if the user is not annoyed, the t-test should indicate that the two sets are similar. We then manually compare the t-test results with the reported user satisfaction. The sensor metric a success if (1) the t-test indicates a difference and the user satisfaction changes, or (2) the t-test indicates similarity and the user satisfaction does not change. False positives occur when the t-test indicates a difference, but the user satisfaction is the same. False negatives occur when the ttest indicates similarity, but the user satisfaction is different.

Out of our twelve potential sensor metrics (maximum, mean, and variance for pupil radius, pupil movement, delta GSR, and force feedback), we develop a set of the six best individual sensor metrics (shown with their respective counts in Table 3.1). The

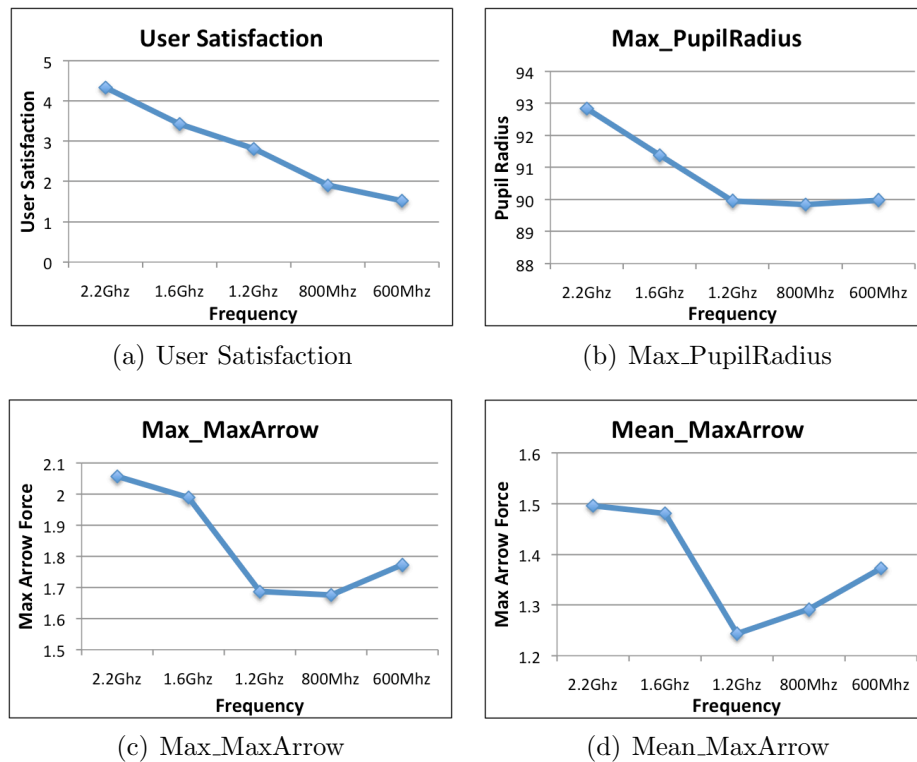


Figure 3.4. Averages of the three best individual sensor metrics and the user satisfaction ratings across all 20 users. The three sensor metrics have a very strong correlation with the reported user rating.

success rates of the six sensor metrics are all above 60% with the top three predicting similar/different user satisfaction with nearly 70% accuracy. The false positive rate ranges from 11.9%–14.3% and the false negative rate ranges from about 15.5%–32.1%¹. These results show that there is a strong correlation between changes in satisfaction and changes in the physiological readings.

¹The false positive rate implies a lost opportunity for reducing frequency, but no reduction in user satisfaction. Assuming that the sensors are independent, combinations of them may be used to reduce the false negative rate. Furthermore, any DVFS algorithm based on these sensors could treat the sensor readings conservatively, reducing the effect of false negatives. In the system we describe in Section 3.4, we use combinations of sensors and evaluate both aggressive and conservative uses of their readings.

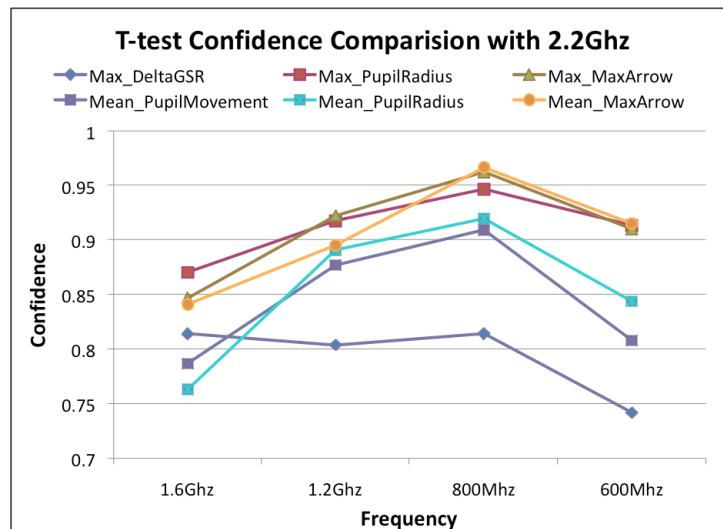


Figure 3.5. The average confidence provided by the t-test between a frequency and the highest frequency across all 20 users and all sensor metrics. A high confidence indicates a difference. As frequency difference increases, the sensor metrics differentiate better, except for the lowest frequency.

To confirm our findings for the entire set of users, we average the sensor metrics across all users and look for trends. Figure 3.4 shows the averaged data for user satisfaction and the top three sensor metrics. There is a clear correlation between our sensor metrics and user satisfaction. For reference, the rest of the raw data is shown in Figure 3.10 in Section 3.7. The sensor metrics exhibit some noise across users but, overall, these results show that a change in user satisfaction generally results in a change in sensor readings. This behavior, together with the high prediction accuracy, shows that user satisfaction and physiological traits are correlated.

We now consider the confidence level reported by the t-test for each comparison. A high confidence level indicates that the two sets of data being tested are different. Figure 3.5 shows the average confidence levels across all users for each comparison. As performance decreases, confidence that the user satisfaction is different tends to increase.

This signifies that the physiological readings differ more at lower performance levels. However, the lowest frequency level does not follow the same trend. We postulate that at this frequency level, the performance is so low that some users stop caring about the game. During the user studies, we recall users complaining about the performance and talking to the proctor instead of change in such situations. Nevertheless, even for this case, the sensor readings show significantly different behavior when compared to the highest frequency.

An important decision we have to make is how to decide when two readings are different. According to our subjective observations, the Need for Speed game exhibits very similar performance at 2.2 GHz and 1.6 GHz, but the performance quickly decreases at lower frequencies. A confidence level of 85% makes this distinction correctly when averaging across all users, and continues to distinguish correctly for a different set of users in the third study. Thus, we adopt an 85% confidence level in the t-tests for the rest of the paper.

In summary, these two initial user studies indicate that:

- a drastic drop in performance results in noticeable changes in our sensor metrics,
- and
- physiological readings can be used to infer user satisfaction.

3.4. Using Physiological Traits for Dynamic Voltage and Frequency Scaling

To demonstrate a use of empathic inputs, we construct a *Physiological Traits-based Power-management* (PTP) system for inferring user satisfaction from physiological readings and driving a DVFS algorithm.

Algorithm 1: PTP training algorithm.

```

1 procedure FIND-SETTLED-FREQ()
2   Frequency:  $f \leftarrow \text{MAX\_FREQ} - 1$ 
3   while  $f$  is in frequency range do
4     if TEST-SAME(MAX_FREQ,  $f$ ) then
5       |  $f \leftarrow f - 1$ 
6     else if Majority vote of 3 calls to TestSame(MAX_FREQ,  $f$ ) is true then
7       |  $f \leftarrow f + 1$ 
8     else
9       | while  $f$  is in frequency range and Majority vote of 3 calls to
10      | TestSame(MAX_FREQ,  $f$ ) is false do
11      | |  $f \leftarrow f + 1$ 
12      | return  $f$ 
13
14
15 procedure TEST-SAME( $f1, f2$ )
16   Collect sensor metrics at  $f1$  for 20 seconds
17   Collect sensor metrics at  $f2$  for 20 seconds
18   t-test each sensor metric at  $f1$  and  $f2$  with confidence level of 85%
19   if more than 50% of sensor differ then
20     | return false
21   return true
22

```

The goal of PTP is to determine the minimum operating frequency that maintains user satisfaction. Specifically, PTP first runs a training phase with the target application (the algorithm for the training phase is detailed in Algorithm 1). PTP begins by comparing sensor readings at the second-highest frequency and the readings at the highest frequency. Each comparison (detailed in the TEST-SAME procedure within Algorithm 1) consists of (1) running for 20 seconds at the highest frequency, (2) running for 20 seconds at the testing frequency, and (3) a t-test between each of the sensor metrics. Initially, the algorithm aims at quickly reducing the frequency, if possible. The algorithm consecutively tests the frequencies for noise in the sensors. If two out of three tests report that the sensor

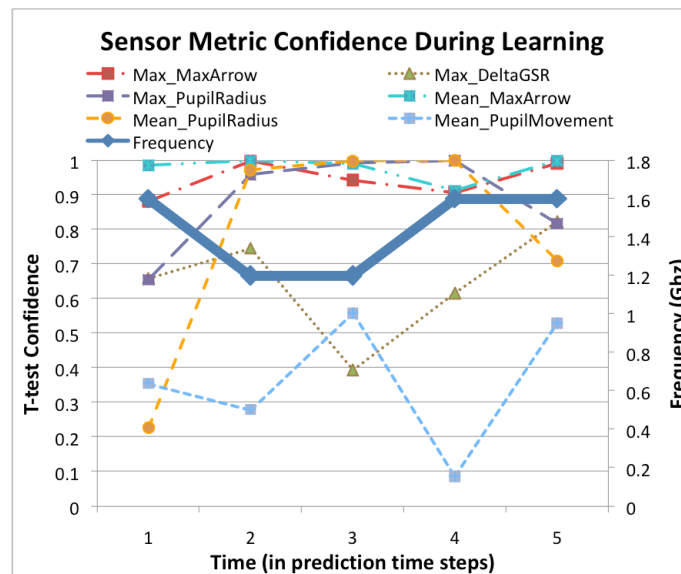


Figure 3.6. Trace of sensor metrics and the frequency during the training phase of the PTP algorithm. When sensor readings are compared for 1.2Ghz and 2.2Ghz, the majority of the sensors result in a high t-test, indicating that the user's state changes. As the algorithm adjusts to test 1.6 GHz, the physiological traits show less change. PTP chooses 1.6Ghz for the rest of the experiment

metrics have changed, the majority vote test concludes that the two frequencies are the different; if not, it reports they result in the same user satisfaction. PTP repeats the majority vote for each frequency until it finds a frequency that does not pass. Then, it starts moving up from this point until it finds the level that passes the majority test. This frequency is called the *settled frequency*. Settled frequency is used as the maximum frequency during the execution of this application (in other words, the operating frequency is never increased to above the settled frequency).

It is important to note that from the users perspective, the training and testing phases are not visible. The user simply interacts with the computer as normal.

An example of the interaction between the sensor metrics and PTP training is shown in Figure 3.6. The figure shows a trace of the algorithm as it settles on a frequency (in this case, 1.6 GHz). The x-axis is time. Each step represents a 40 second period: 20 seconds at the highest frequency, and 20 seconds at the test frequency. The bold line with diamonds shows the test frequency, corresponding to the right vertical axis. The confidence levels of the t-tests for each sensor metric is shown in each time step, with the confidence indicated by the left vertical axis. A confidence above 85% indicates that the sensor metric differs between the two frequencies. We begin at 1.6 GHz. At this point, only 2 of the 6 sensors are different so we continue down to 1.2 GHz. At 1.2 GHz, there is a large change in Mean_PupilRadius. In fact, Max_MaxArrow, Mean_PupilRadius, Mean_MaxArrow, and Max_PupilRadius all exhibit high confidence for two tests and therefore reject the majority vote test for 1.2 GHz. The frequency increases to 1.6 GHz, and the sensor metrics return to values indicating that the sensors are the same, therefore predicting the user is satisfied. The algorithm settles at this frequency.

The PTP control algorithm is orthogonal to most other DVFS strategies. Although PTP provides a long-term prediction of user satisfaction, another DVFS strategy can be used for short-term decisions. We build PTP on top of an *Adaptive* DVFS strategy that is based upon the Linux ondemand DVFS governor [81]. This strategy is described in Algorithm 2. In short, if utilization increases above UP_THRESHOLD, the frequency increases to the maximum frequency. If the utilization is below the DOWN_THRESHOLD, the algorithm finds the frequency that maintains above 80% utilization. We use 200 ms for both UP_DELAY and DOWN_DELAY, 80% for UP_THRESHOLD and 30% for the DOWN_THRESHOLD.

Algorithm 2: Linux ondemand governor algorithm.

```

1 for every CPU in the system do
2   if UP_DELAY milliseconds since last check then
3     if utilization > UP_THRESHOLD then
4       | increase frequency to maximum
5     |
6   if DOWN_DELAY milliseconds since last check then
7     if utilization < DOWN_THRESHOLD then
8       | decrease to lowest frequency that keeps the utilization at 80%
9     |
10  |
11  |

```

PTP uses the minimum value of the frequency provided by the PTP control policy and the *Adaptive* control policy. Although the idea of combining the DVFS schemes may seem simple, there are benefits to such a solution. For example, a burst of keyboard or mouse events often cause adaptive DVFS control schemes (e.g., Windows XP DVFS [76] or the Linux ondemand control policy [81]) to unnecessarily raise the frequency to the maximum level. PTP prevents this by limiting frequency at the minimum level necessary to satisfy the user. In other words, PTP allows an adaptive DVFS scheme to make better short-term decisions when the CPU utilization is generally low. For applications that satisfy the user at high utilization, PTP may set the frequency to a lower level (if it predicts that the user is satisfied with that level), saving a significant amount of power.

Ideally, we would like to explore the combinations of sensor metrics for users and applications as well as search the parameter space for the PTP thresholds, but this would require real users in the loop and therefore be slow. A single user study with three applications takes about an hour of experimental lab time, not including the time to schedule the experiment. Therefore, trying multiple combinations quickly becomes very

time consuming. We settled on the six most accurate individual sensor metrics listed in Table 3.1 and close the loop for evaluation with user studies.

Picking one set of sensor metrics opens some questions. Will the sensor metrics generalize across applications? Even for a single application, how does the sensitivity depend on users? By using the same set of sensor metrics across all users and applications, it is very possible that we will occasionally annoy some users. To increase the sensitivity to our experiments, we develop two variations of PTP: an aggressive PTP (*a*PTP) and a conservative PTP (*c*PTP). *a*PTP operates exactly as the PTP algorithm described in this section. *c*PTP is similar to *a*PTP but selects the frequency level one step higher than *a*PTP

3.4.1. Implementation, Integration, and Deployment

The PTP system is implemented as a user-space program that executes before each application run in the user studies. Data from the biometric devices are collected on a separate workstation and sent to the experimental laptop via a TCP socket connection. In production systems, we envision biometric input devices being managed by the operating system like traditional input devices. We have designed PTP as a proof of concept for using biometric input devices to improve architecture-level decisions. Other approaches to using biometric data different from ours could potentially lead to even stronger results. Here, we are concerned with providing the first evidence of the clear benefits of using biometric data in architecture-level decision making.

In a real-world implementation, the power consumption of the biometric devices would need to be outweighed by the power savings due to the PTP. The sensors chosen for this work all conform to this requirement. Piezoresistive force sensors may be measured with very little additional energy using a voltage-divider circuit and an analog-to-digital converter, which are both common, low-power circuits. GSR is also a simple resistive measurement, and requires only a voltage divider and an analog-to-digital converter. An eye tracker requires an infrared camera, infrared LEDs, and the capacity for image processing. Collectively, the eye tracker sensor could operate on well below a Watt [117, 62]. Although some of these sensors may be expensive today, the technology for producing sensors capable of operating within desirable power constraints and at a low cost has already been developed. Additionally, the processing needs to interpret the sensors could also be assigned to a core of a chip multiprocessor, reducing the additional hardware required.

3.5. Experimental Results

In this section, we evaluate the *a*PTP and *c*PTP systems. We compare both PTP variants with the *Adaptive* scheme described in Section 5. We use the Need for Speed (NFS), Tetris, and Word applications and 20 users. In each run of an application, we begin with the training phase described in Section 3.4. The training phase varies based upon the number of majority vote tests performed by the PTP strategy. Afterwards, the user continues to use the *Adaptive* scheme and the *a*PTP scheme for 2.5 minutes each. The order of the *a*PTP and the *Adaptive* scheme is randomized between experiments. The last 10

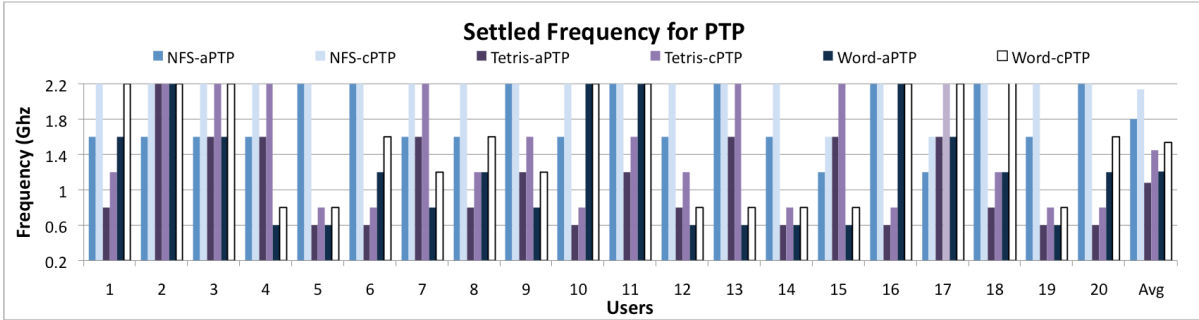


Figure 3.7. Frequency that *a*PTP and *c*PTP settle on for the Need for Speed, Tetris, and Word applications. *c*PTP for Word is omitted because it results in very little change in power savings and user satisfaction.

users subsequently use the *c*PTP scheme for 2.5 minutes. At the end of each run, the user is asked to verbally report satisfaction based upon the scale described in Section 3.3.2.

During experiments, we capture traces of the frequency. A National Instruments 6034E data acquisition card measures the potential drop across a low-impedance resistor in series with the laptop power cable. This allows us to measure the system power consumption as frequency traces are replayed. The total system power includes the power consumed by the fully-operating laptop including the processor, a fully-lit 15.1” laptop display, network interface, and other peripherals. The take-away points from our evaluation are:

- User satisfaction for *a*PTP and *c*PTP are nearly identical to the underlying *Adaptive* scheme, and
- *a*PTP and *c*PTP save 18.4% and 11.4% total system power, respectively.

3.5.1. User Satisfaction and Power Savings

In Figure 3.7, we present the frequencies that *a*PTP and *c*PTP settle on for NFS, Tetris, and Word. The x-axis corresponds to the users and the y-axis is the settled frequency. Each cluster shows the settled frequency for both PTP variants and all applications.

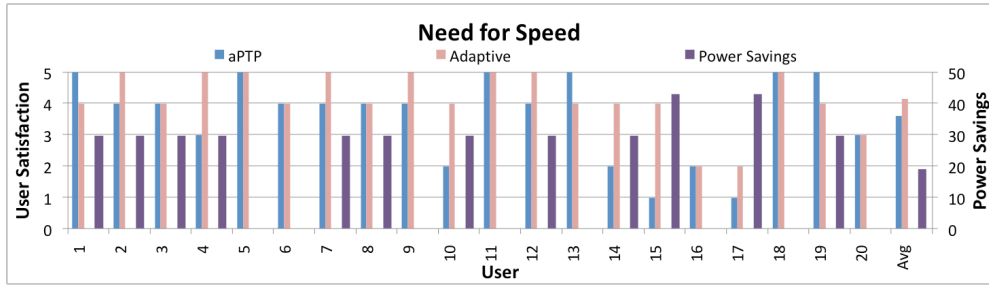
NFS is a CPU-intensive application for which observable performance is sensitive to CPU frequency. *aPTP* picked either 1.6 GHz or 2.2 GHz for 18 out of the 20 users. This is drastically different from Tetris, where the observable performance is less sensitive to CPU frequency. The average frequency chosen by *aPTP* for Tetris is 1.08 GHz. Similarly, for Word, the average frequency chosen is 1.2 GHz. This clearly demonstrates *aPTP*'s ability to intelligently detect the cases where CPU frequency can be lowered. Since for the Tetris and Word application, the lower frequencies and higher frequencies result in similar physiological responses, *aPTP* lowers the frequency. As indicated by user satisfaction levels, this achieves significantly higher efficiency without causing any dissatisfaction. Note that a user-specific customization is achieved purely based on the physiological readings from the users, without explicit input or knowledge of program phase.

There are some cases in Tetris and Word (14 out of 40 cases altogether), where a higher frequency of 1.6 GHz or 2.2 GHz is picked by *aPTP*. We checked the logs of physiological readings and found that the eye tracking data was missing in 4 of these 14 cases. This occurs when the user shifts in a manner such that pupil is not captured by the eye tracker camera. This introduces significant noise to the decision making system and results in a higher frequency being chosen. Another 3 cases correspond to self-admittedly inexperienced users. These users show erratic behavior. Thus, the sensor readings are noisy and our system conservatively sets the frequency at a high level. We must note that, although this looks like a lost opportunity for power saving, it is an interesting feature of the overall scheme: if for one reason or another, the sensor readings become noisy, our system conservatively sets the maximum allowed frequency to a high one, thereby avoiding false negatives (i.e., cases where the user is dissatisfied and our system predicts

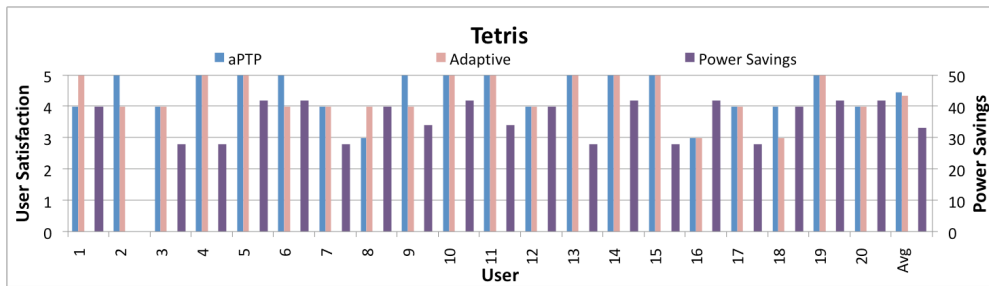
them to be otherwise). For Word, we are limited to utilizing only 4 metrics, compared to the 6 used in NFS and Tetris, because Max_MaxArrow and Mean_MaxArrow cannot be used (the user does not press the arrow keys often). Nevertheless, with Word, *a*PTP succeeds in picking low CPU frequencies (1.2 GHz and below) for 13 out of the 18 users with valid sensor readings. Similarly, for Tetris, *a*PTP picks a low frequency for 13 out of 15 users with valid sensor readings.

The reported user satisfaction ratings and power savings for each of the applications comparing *a*PTP and the *Adaptive* scheme are presented in Figure 3.8. The figure shows clustered bars for each user. The left two bars in each cluster represent the user satisfaction with *a*PTP and with the *Adaptive* scheme and correspond to the leftmost vertical axis. The right bar in each cluster represents the total power savings corresponding to the vertical axis on the right. For our two CPU-intensive applications, *a*PTP saves a considerable amount of total power. On average, for NFS (presented in Figure 3.8(a)), *a*PTP reduces power consumption by 19.2%, and for Tetris (presented in Figure 3.8(b)), *a*PTP reduces total power consumption by 33.3%. Word (presented in Figure 3.8(c)) is only CPU-intensive in short bursts and *a*PTP only saves 1.7% system power. For both Tetris and Word, *a*PTP also does not impact user satisfaction. However for NFS, *a*PTP trades off a small amount of user satisfaction for power savings. For this application, *a*PTP is too aggressive for some users. Averaged across three applications, *a*PTP saves 18.4% system power when compared to the *Adaptive* scheme.

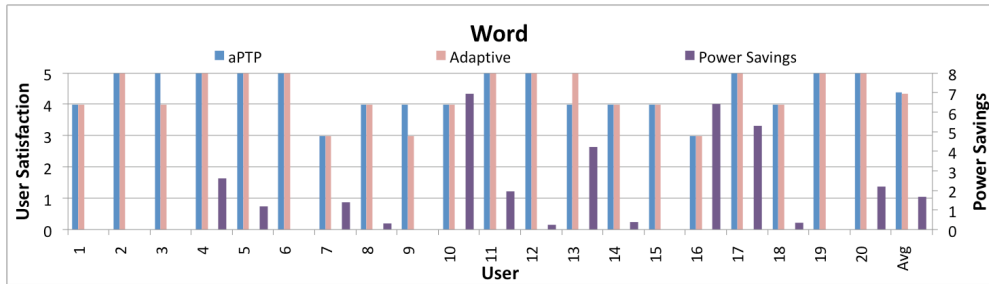
To explore a more conservative PTP scheme, we evaluate *c*PTP with 10 users. Figure 3.9 presents the results of this study. The graph is in the same format as Figure 3.8. By using *c*PTP, we trade off improved user satisfaction with power savings. *c*PTP tends



(a) Need for Speed



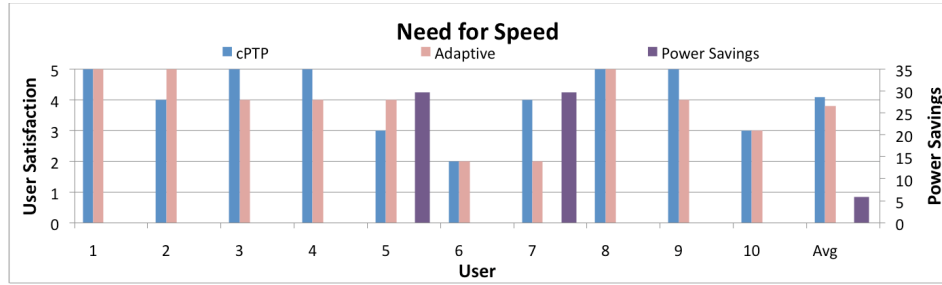
(b) Tetris.



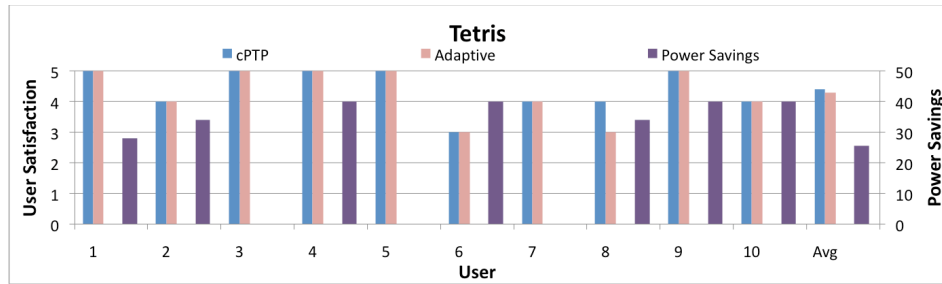
(c) Microsoft Word.

Figure 3.8. User satisfaction and power consumption for the Need for Speed, Tetris, and Word applications. The left two bars per cluster show the user satisfaction for *a*PTP and the Adaptive DVFS schemes. The right bar in each cluster shows the total system power savings.

to maintain the highest frequency for NFS and saves 5.9% system power, while maintaining the same satisfaction level as the *Adaptive* scheme. *c*PTP trades off the decreased power savings with an improved average user satisfaction rating compared to *a*PTP. *c*PTP also maintains a high user satisfaction for Tetris, and the power savings drop from 33.3%



(a) Need for Speed



(b) Tetris.

Figure 3.9. User satisfaction and power consumption of *c*PTP for the Need for Speed and Tetris applications. Word is not included because power savings and user satisfaction levels are nearly identical to *a*PTP. The left two bars per cluster show the user satisfaction of *c*PTP and the Adaptive DVFS schemes. The right bar in each cluster shows the total system power savings. Using *c*PTP, we trade-off a decreased power savings with improving user satisfaction when compared to *a*PTP.

to 25.6%. Averaged across three applications, *c*PTP saves 11.4% system power while maintaining the user satisfaction.

Overall, our results are very encouraging: they show that PTP can successfully sense physiological traits, predict user satisfaction, and drive a DVFS scheme that saves considerable power while maintaining user satisfaction.

3.6. Summary

In this chapter, we made a case for the addition of new input devices that provide information on human state in future computer architectures. Specifically, we explored the use of three biometric sensors: an eye tracker to measure pupil dilation and pupil movement, a galvanic skin response sensor for sensing user arousal, and force sensors on the keyboard for sensing behavioral traits. We have conducted multiple user studies. The first showed that human physiological readings do in fact change with changes in performance. The second shows that biometric readings are correlated with user satisfaction. Based upon the observations in these initial studies, we constructed a Physiological Traits-based Power management (PTP) system for driving dynamic voltage and frequency scaling on a processor. PTP was designed to be orthogonal to most other DVFS techniques. We built our system in combination with an adaptive DVFS scheme based on the Linux ondemand governor. An evaluation using an additional user study showed that an aggressive PTP scheme reduced the total system power consumption of the laptop by up to 33.3% for an application averaged across users (18.1% averaged across three applications), while a conservative PTP scheme reduced the total system power consumption by up to 25.6% across users (11.4% averaged across three applications). Overall, these results show that a robust system can be built that makes decisions based upon observing biometrics sensors. This demonstrates the potential for incorporating biometric information into the architecturelevel decision making process.

3.7. Raw Data from Motivational User Study

This section expands upon discussion in Section 3.3.2. Figure 3.10 presents the raw data for six of the sensor metrics. The results for each user is presented in a row in the table of graphs and each column corresponds to a different sensor metric (the first column presents the reported user satisfaction level). In each of the graphs, the x-axis represents the frequency with 1 being the highest (2.2 Ghz) and 5 being the lowest frequency (600 Mhz). The y-axis represents the user satisfaction rating for the first column and the mean of the sensor readings for the remaining columns. The raw data shows that the sensor metrics are can be noisy. However, in general, a change in the user satisfaction is reflected by a change in sensor metrics. If we consider the average behavior (presented in the last row), we see that most sensors show a strong relation to the user satisfaction levels.

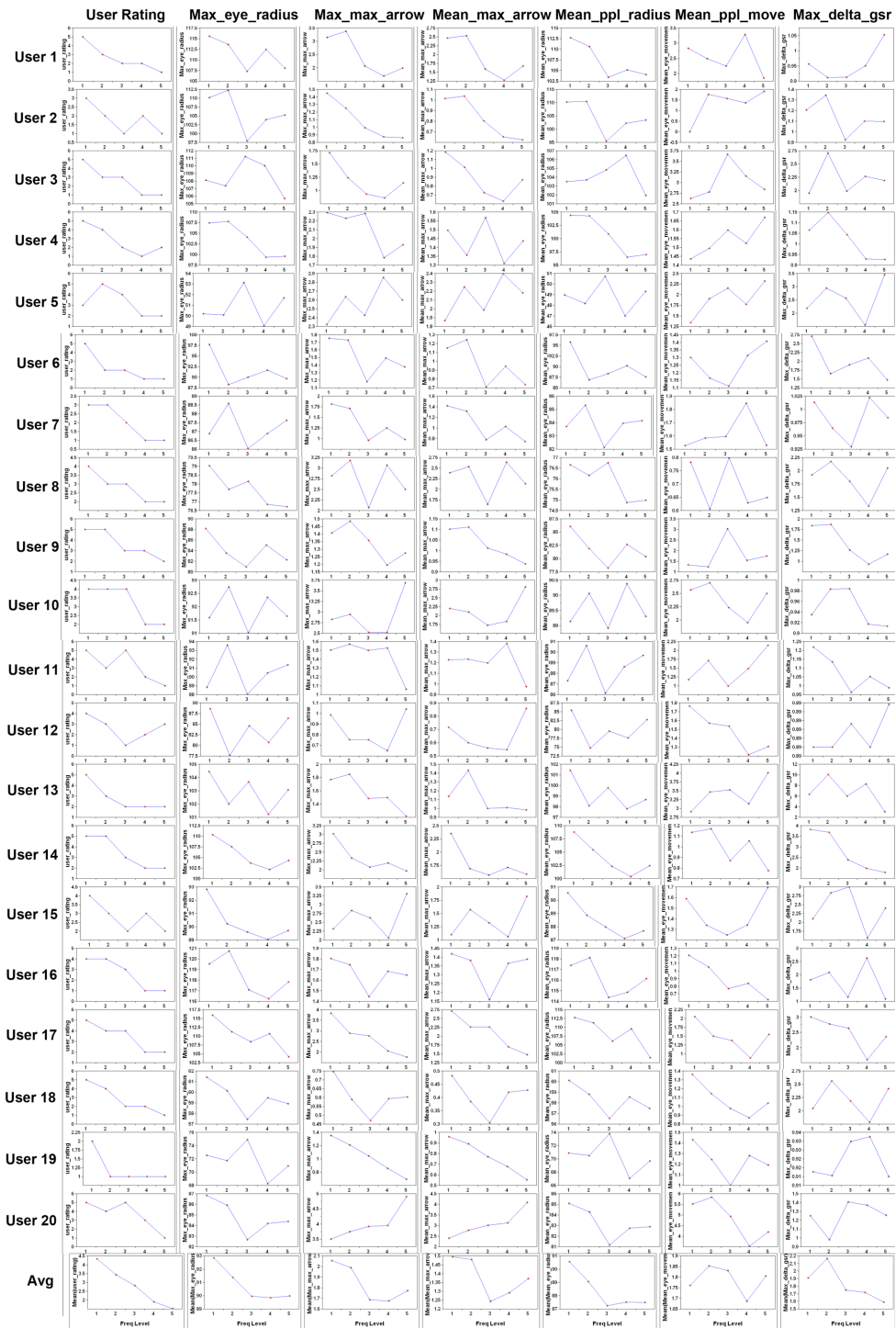


Figure 3.10. Physiological traits and user satisfaction when randomly changing to multiple frequencies at different points in Need for Speed.

CHAPTER 4

User Activity: Studying User Behavior to Drive Optimization

In recent years, there has been a tremendous shift in the market for personal computing. Users are in the midst of a mass transition from stationary desktop computers to a mix of mobile architectures, e.g., PDAs, cellular phones, media players, and netbooks. Portable music players are now ubiquitous with nearly four in ten Americans owning a portable MP3 player [9]. Mobile phones have been adopted faster than any technology in history [35]. Netbooks are projected to follow a similar trend in the coming years [34]. Although mobile architectures provide the convenience of portable computation, entertainment, and communication, their utility is severely constrained by their battery life. As the demand for mobile architectures continues to grow, it will become increasingly important for architects to focus on understanding and optimizing the power consumption of these energy-constrained architectures.

Power estimation and optimization has been a popular area of research in embedded and mobile architectures for many years. Researchers have studied power at many levels, including the circuit, architecture, and system levels. However, there is one critical factor that architects have largely ignored – the end user. On a mobile architecture, the end user is the workload: mobile architectures typically run applications which interact directly with them. Execution of batch jobs and long-running services are minimized, or even disallowed, as on the iPhone [8]. As a result, the usage behavior of the end user drives execution, which in turn, determines the power consumption. Architects should treat

the end user as the workload, and study trends, properties, and patterns in user activity. Without understanding these patterns, it is not possible to clearly understand the impact of any optimization on user experience or the real device power consumption.

In this chapter, we describe tools and methods for studying the power consumption of real mobile architectures with respect to user activity. We develop a logger application for Android G1 mobile phones that logs user activity and sends traces back to our servers. We release the logger into the wild to collect traces of real users on real mobile devices in real environments. We then demonstrate how the traces can be used to characterize power consumption, and guide optimizations on mobile architectures.

We present an regression-based power estimation model which uses high-level system measurements to estimate power consumption. The measurements used as inputs are chosen to be representative of underlying hardware components. Furthermore, the measurements are easily accessible and can be collected by our logger (which operates entirely in user space). We develop the model using in-house power measurements and show that the power estimation model can accurately predict the power consumption by validating the model using a separate device and random workloads.

We then use our power model to characterize the power consumption of the Android G1. We first analyze a set of synthetic testing workloads, and find that the breakdown of power consumption among hardware components varies significantly based upon the workload. Our findings motivate the need for understanding real workloads in real environments. We then analyze the traces from our 20 real users. Again, we find a large variation in the power breakdown between users. Averaged across our users, our results also indicate that the CPU and the screen are the two most power-consuming components.

Finally, we demonstrate an example of studying user activity patterns to guide the development of novel power optimizations. We study active screen behavior and observe that the majority of active screen time is dominated by a relatively small number of long active screen intervals. Thus, optimizing for long screen intervals would be profitable for reducing power consumption. Targeting these long intervals enables us to develop a novel scheme that utilizes *change blindness*. Change blindness refers to the inability of humans to notice large changes in their environments, especially if the changes occur in small increments. We implement optimizations that slowly decrease CPU frequency and screen brightness during long active screen intervals. We conduct a user study testing these schemes and show that users are more satisfied with a system that slowly reduces the screen brightness rather than abruptly doing so, even though the two schemes reach the same brightness level. Overall, our schemes save 10.6% of the phone energy consumption on average with minimal impact on user satisfaction.

Overall, we make the following contributions:

- We develop an accurate linear-regression-based power estimation model which leverages easily-accessible measurements to accurately predict the system-wide power consumption of a mobile architecture;
- We use our power estimation model to characterize the power consumption of an Android G1 mobile architecture with respect to user activity patterns;
- We demonstrate an example of developing optimizations for CPU frequency scaling and screen brightness based upon user activity patterns; and
- We utilize change blindness for power optimization during active use.

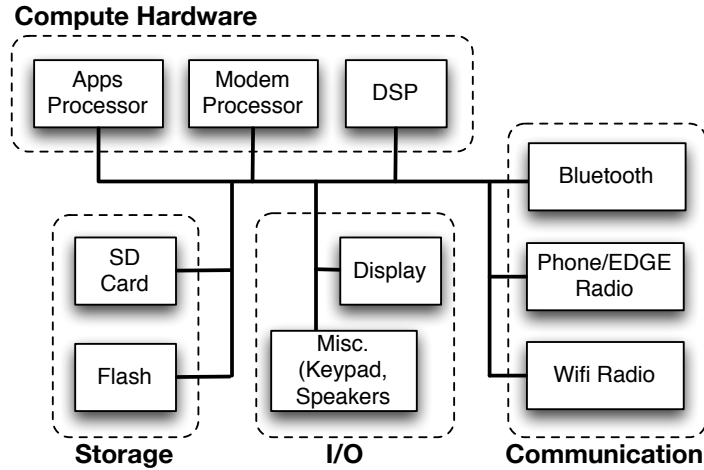


Figure 4.1. High-level overview of the target mobile architecture.

The rest of the chapter proceeds as follows. In Section 4.1, we discuss our experimental setup. Section 4.2 presents our linear-regression-based power estimation model. In Section 4.3, we study user activity traces to derive the power breakdown on real mobile phones and identify a potential optimization direction by studying screen activity. Section 4.4 presents optimizations for the CPU and screen that leverages change blindness. We evaluate the optimizations in Section 4.5. We conclude in Section 4.6.

4.1. Experimental Setup

Our target mobile architecture in this paper is the HTC Dream, a smartphone developed by HTC that supports the open source Google Android mobile device platform [46]. Although we focus on a specific mobile architecture for experimentation, our contributions and findings could easily extend to other mobile architectures.

We use the G1 Android Developer Phone 1 (ADP1), a rooted and SIM-unlocked version of the HTC Dream. We use the Android OS 1.0 stock system image for the ADP1

and develop with the Android 1.0 SDK. The Android platform consists of a slightly modified 2.6.25 Linux kernel, and a general framework of C, C++, and Java code. The framework includes the Dalvik Virtual Machine (VM), a variant of Java implemented by Google. All userspace applications are Dalvik executables that run in an instance of the Dalvik VM.

A high-level diagram of the mobile architecture is shown in Figure 4.1. The ADP1 has a 3.2 inch HVGA 65K color capacitive touch screen, uses a Qualcomm MSM7201A chipset, and a 1150 mAh lithium-ion battery [113]. The Qualcomm MSM7201A chipset contains a 528 MHz ARM 11 apps processor, an ARM 9 modem processor, a 528 MHz ARM 11 Jazelle Java hardware acceleration, QDSP4000 and QDSP5000 high-performance digital signal processors, quadband GPRS and EDGE network, integrated Bluetooth, and Wi-fi support.

To the best of our understanding, the ARM 11 apps processor runs the Android platform and executes the applications on the device. It is rated at 528 MHz and supports dynamic frequency scaling (DFS), but is scaled down in the platform to run at 124 MHz, 246 MHz, and 384 MHz. The highest frequency of 528 MHz is not used. The ARM 9 modem processor is a separate processor that runs a proprietary operating system and is in charge of the communications of the phone. The Jazelle Java hardware acceleration processor is not used as the Android platform runs Dalvik executables which are not fully compatible.

We build our power estimation model using real power measurements. We instrument the contact between the phone and the battery and measure the current with a Fluke i30 AC/DC current clamp. We use the OS reported battery voltage as the operating

battery voltage. The linear regression model is created using the R Statistical Computing Environment [86].

We develop a logger application that logs system performance metrics and user activity. The logger runs as a Dalvik executable. It does not require any special hardware or OS support, and runs on consumer HTC Dream devices, such as the T-Mobile G1 phone. The logger periodically looks for a network connection and sends the logs back to our server. All data is anonymous by the time it reaches our server.

To obtain users for our study, we publicized our project for a month on multiple university campuses, as well as to the general public. Users install the logger through the Android Market. To minimize potential bias in our data, all volunteers remain anonymous. Volunteers are notified that we do not collect any data that could be used to identify them. We also provide a complete list of collected data to maintain transparency with the users. To avoid any change in user behavior, the logger application is designed to be as unintrusive as possible. It automatically starts upon installation or after the boot process, and consumes minimal system resources.

For the data in this paper, we use the logs from the 20 users who have the largest logged activity. The cumulative log data represents approximately 250 days of real user activity. To explore usage patterns when the mobile device is battery-constrained, we focus on time intervals when the battery is not charging. From all of the logs, we extract 860 time intervals where the battery is not charging, which add up to a total of 145 days of user activity.

4.2. Power Estimation Model

We now discuss our approach to system-level power estimation for mobile architectures. We model the architecture having two distinct power states:

Active: : The apps processor is operational. This occurs during active usage when the screen is on, or if a system wake lock is held to ensure that the apps processor remains on while the screen is off.

Idle: : The device is in a low-power sleep mode. The apps processor is not operational but the modem processor is still active (also called "Standby" mode).

The power consumed in the Idle state is significantly lower than the Active state, and is relatively invariant under typical circumstances (measured to be around 70 mW). In contrast, power consumed in the Active state is considerably higher (300~2000+ mW), and varies significantly by workload. Our power estimation model focuses primarily on modeling Active state power consumption.

We build our power estimation model based on high-level measurements collected for a set of the hardware units. We choose a linear regression method to build the model. Linear regression fits an output variable to a set of independent input parameters by corresponding linear coefficients.

4.2.1. Choosing Parameters

Table 4.1 lists the parameters selected for the power estimation model, including the final coefficients used in our power estimation model. We model most of the hardware components on the ADP1, including the CPU, screen, calls, EDGE/Wi-fi network, SD card, and the DSP processor.

HW Unit	Parameter	Description	Range (of $\beta_{i,j}$)	Coefficient (c_j)	units
CPU	hi_CPU_util	Average CPU utilization while operating at 384 MHz	0–100	3.97	$mW/\%$
	med_CPU_util	Average CPU utilization while operating at 246 MHz	0–100	2.79	$mW/\%$
Screen	screen_on	Fraction of the time interval with the screen on	0–1	150.31	mW
	brightness	Screen brightness	0–255	2.07	$mW/(\text{step})$
Call	call_ringing	Fraction of the time interval where the phone is ringing	0–1	761.70	mW
	call_off_hook	Fraction of time interval during a phone call	0–1	389.97	mW
EDGE	edge_has_traffic	Fraction of time interval where there is EDGE traffic	0–1	522.67	mW
	edge_traffic	Number of bytes transferred with the EDGE network during time interval	≥ 0	3.47	mW/byte
Wifi	wifi_on	Fraction of time interval Wifi connection is on	0–1	1.77	mW
	wifi_has_traffic	Fraction of time interval where there is Wifi traffic	0–1	658.93	mW
	wifi_traffic	Count of bytes transferred with Wifi during interval	≥ 0	0.518	mW/byte
SD Card	sdcard_traffic	Number of sectors transferred to/from Micro SD card	≥ 0	0.0324	mW/sector
DSP	music_on	Fraction of time interval music is on	0–1	275.65	mW
System	system_on	Fraction of time interval phone is not idle	0–1	169.08	mW

Table 4.1. Parameters used for linear regression in our power estimation model.

CPU: : The CPU refers to the apps processor and supports DFS between three frequencies, as described in Section 4.1. The lowest frequency is never used on consumer versions of the phone, and is too slow to perform basic tasks. Thus, only the high (384 MHz) and medium (246 MHz) frequencies are considered in our model.

Screen: : The screen parameters include a constant offset indicating whether the screen is on and a second parameter to model the effect of the screen brightness.

Call: : We model the power during phone calls by measuring the time spent ringing and the duration of the phone calls.

EDGE: : The EDGE network power consumption parameters consider whether there is any traffic and the number of bytes of traffic during a particular time interval.

Wi-fi: : The Wi-fi power consumption is modeled similar to the EDGE network but also includes a parameter for whether Wi-fi connectivity exists.

SD Card: : We consider the number of sectors transferred per time interval.

DSP: : We model the DSP by checking an internal variable within the Android SDK for whether there is a multimedia file playing. This variable is on during music and video playback.

System: : The power that is not accounted for with the hardware components listed above are put into a catch-all variable that we simply refer to as the miscellaneous **System** power in Table 4.1. The System power corresponds to the constant y-intercept in a linear regression model, or k , as it will be described in the following section.

4.2.2. Building the Estimation Model

To develop our model, we use real-time measurements of our target phone. The overall idea is to find the relationship between the collected system statistics and the power consumption. Hence, the input to our model is the statistics collected from the phone. The output is the total power consumption. During training, we provide the measured power consumption and use the R-tool to build the linear regression model. Specifically, during training, we have performed a series of tasks to stress different components of the hardware. During these tests, we (1) measure the real-time power consumption of the phone and (2) collect statistics about the chosen parameters described in the previous section. The raw data samples are collected every second for the synchronous data (e.g., CPU utilization). We sample at 1 Hz to reduce perturbation on the system and minimize the execution and power consumption of the logger. During training, the collected statistics and the measured power consumption levels are fed into the R-tool to find the regression

coefficient c_j for each parameter. Once a model is generated, one can predict the power consumption by simply providing the statistics for the selected parameters.

As we will discuss in Section 4.2.3, this approach results in a highly accurate power model. In addition, it can be used to estimate the power consumption of each individual hardware component. If a single measurement (e.g., the value for screen brightness) in sample i is $\beta_{i,j}$, then the power $p_{i,j}$ contributed by the corresponding hardware component for the parameter coefficient c_j is:

$$(4.1) \quad p_{i,j} = \beta_{i,j} \cdot c_j$$

Power not attributable to any available measurement is aggregated into a constant offset k . The total system power P_i for sample i with n available measurements is then modeled with the sum of these n power values:

$$(4.2) \quad P_i = k + (p_{i,0} + p_{i,1} + \dots + p_{i,n})$$

$$(4.3) \quad = k + ((\beta_{i,0} \cdot c_0) + (\beta_{i,1} \cdot c_1) + \dots + (\beta_{i,n} \cdot c_n))$$

Allowing $x_i = (\beta_{i,0}, \beta_{i,1}, \dots, \beta_{i,n})$ for each sample i and $c = (c_0, c_1, \dots, c_n)$ reduces this to:

$$(4.4) \quad P_i = k + x_i \cdot c$$

Taken across m samples, this model takes the form:

$$(4.5) \quad \begin{pmatrix} P_0 \\ P_1 \\ \vdots \\ P_m \end{pmatrix} = k \cdot \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix} + \begin{bmatrix} \beta_{0,0} & \cdots & \beta_{0,n} \\ \beta_{1,0} & \cdots & \beta_{1,n} \\ \vdots & \ddots & \vdots \\ \beta_{m,0} & \cdots & \beta_{m,n} \end{bmatrix} \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{pmatrix}$$

$$\text{Letting } P = \begin{pmatrix} P_0 \\ \vdots \\ P_m \end{pmatrix}, X = \begin{bmatrix} \beta_{0,0} & \cdots & \beta_{0,n} \\ \beta_{1,0} & \cdots & \beta_{1,n} \\ \vdots & \ddots & \vdots \\ \beta_{m,0} & \cdots & \beta_{m,n} \end{bmatrix} \text{ and } e = \begin{pmatrix} 1 & \cdots & 1 \end{pmatrix}^T \text{ yields:}$$

$$(4.6) \quad P = k \cdot e + Xc$$

Once values for k and c have been determined, any sample of system measurements x_i may be used to approximate the power consumed by the whole system P_i at the time of the sample with Equation 4.4, and the power contributed by each hardware component during the sample may be approximated using its corresponding measurement in Equation 4.1.

Additionally, the total energy E consumed by the system across a set of such samples X with sampling period t_s may be approximated by the sum:

$$(4.7) \quad E = t_s \cdot \text{sum}(P) = \sum_{i=0}^m t_s \cdot P_i = t_s \sum_{i=0}^m (k + x_i \cdot c)$$

When the phone is in the Idle state, a constant power value ($p_{idle} \approx 68.3$ mW) is used to approximate power consumption. The `system_on` ratio from Table 4.1 indicates the portion of time the system is in the Active state as a ratio between 0 and 1. Thus, when

a log contains both the Active and Idle states, power consumption for a single sample i is modeled as:

$$(4.8) \quad \text{Power}_i = \text{system_on} \cdot (P_i) + (1 - \text{system_on}) \cdot (p_{idle})$$

When the system is in Active state, the power is approximated by the linear regression model P_i ; in Idle state, p_{idle} is used as the approximation. In the linear regression model for Active power, k represents the coefficient for `system_on`.

4.2.3. Validating the Power Model

We approximate the values of offset k and the coefficient vector c using a set of sampled system measurements X with experimentally measured power consumption \hat{P} . Samples are taken from varying workloads to cover the spectrum of possible use scenarios. From Equation 4.6 in Section 4.2.2, this produces the linear equation $\hat{P} = k \cdot e + Xc$, solving for an approximation of k and c . The approximations are shown in Table 4.1 (k is represented by the coefficient for `system_on`; other values in the column jointly form the vector c).

To demonstrate the accuracy of the model, we collect additional logs of system measurements and power consumption. In addition, we collect this set of logs on a separate ADP1 device to ensure that our power estimation model generalizes beyond the device used for training the model. We collect two types of logs. The first type targets specific hardware components of the phone. We name these logs `Runi_Unit`. For example, `Run1_CPU` corresponds to a log with varying CPU utilization. These logs are used for training our power model. The second type of log corresponds to a scenario, or a mix of usage behavior, that stresses multiple hardware components. We name these logs `Scenarioi`.

As an example, `Scenario2` simulates a user listening to music while browsing the web, and then answering a phone call. These logs are not used during training and used to analyze the accuracy of our model for workloads that are not part of the training set. Each log is approximately 5 minutes long.

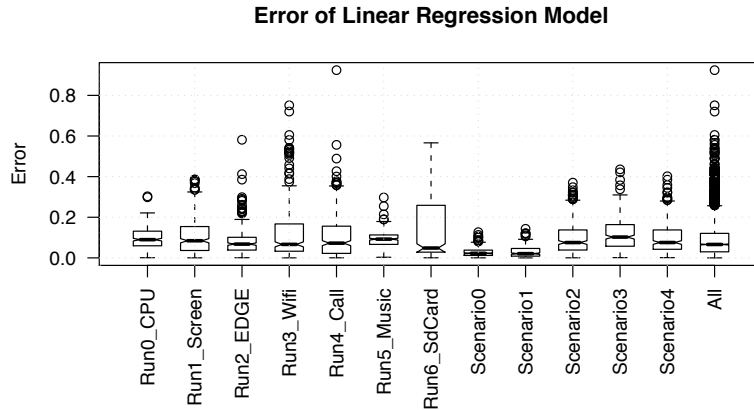
We use this set of logs from a separate mobile device to approximate the error of our power estimation model. Equation (4.4) in Section 4.2.2 provides a power estimation for each sample i . The error considered is the percent absolute relative error ($error_i$) and the percent relative error ($error_j$):

$$(4.9) \quad error_i = \left| \frac{actual - estimated}{actual} \right| = 100 \cdot \left| \frac{\hat{P}_i - P_i}{\hat{P}_i} \right| \%$$

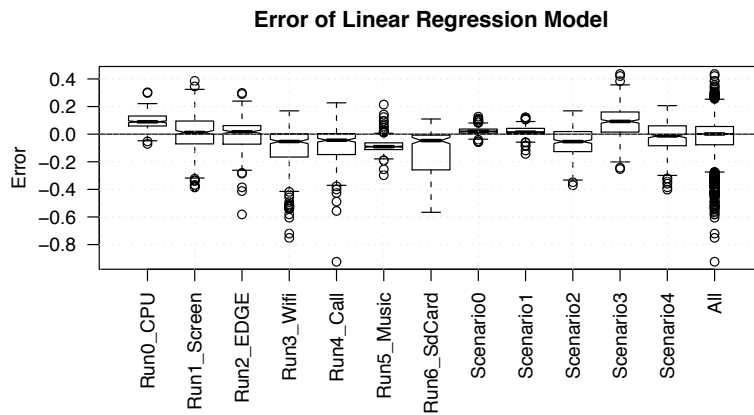
$$(4.10) \quad error_j = \frac{actual - estimated}{actual} = 100 \cdot \frac{\hat{P}_i - P_i}{\hat{P}_i} \%$$

Figure 4.2 presents the range of errors for each of the logs collected, including the logs used in training and the scenario-based logs used for validation. In the figures, the median error for each set is a bold line, the boxes extend to 25% and 75% quartiles, the whiskers extend to the most extreme sample point within $1.5 \times$ the interquartile range, and outliers are independent points. Figure 4.2(a) shows the absolute relative error and Figure 4.2(b) shows the relative error.

Our results indicate that the power estimation model accurately predicts the system-level power consumption of the logs, even though a separate mobile device is used. The median absolute relative error across all of the samples is 6.6%. The median relative error rate is $< 0.1\%$. The hardware-specific logs demonstrate the accuracy of predicting the



(a) Absolute relative error.



(b) Relative error.

Figure 4.2. Error of logger when building the power estimation model on one ADP1 and validating with logs from another ADP1 device.

power consumption of specific hardware components. In general, the model predicts the CPU, EDGE, and music with a low median error rate, and a low variance in the error rates. The error rates in the screen, Wi-fi, phone call, and SD card show a higher amount of variance; but their power consumption can still be predicted accurately with a low median error rate. The scenario-based logs demonstrate that our power estimation model

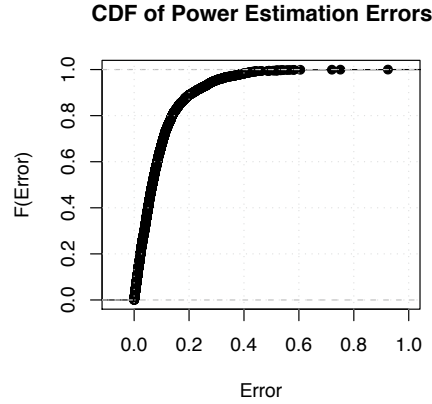


Figure 4.3. Cumulative distribution of power estimation error.

also extends to workloads that combine multiple hardware components, with median errors similar to the hardware-specific logs. Figure 4.3 shows the cumulative distribution of the sample errors. Each (x, y) point represents the ratio of samples (y) at or below a particular absolute relative error (x). 65% of the individual samples are approximated by the model to within 10% absolute relative error, and 90% of the samples are within 20%.

The errors shown in Figure 4.2 are for estimating the average power consumption of individual 4 second time windows. However, it does not provide any indication of total energy estimation across an entire trace. In Figure 4.4, we present the error when comparing the total energy between the power readings, and the results of our power estimation model. Over all of the logs, we achieve $< 0.1\%$ mean error.

4.2.4. Per-Component Power Consumption

With an accurate power estimation model, the combined system power P_i may easily be approximated for any sample i . Furthermore, since this approximation is a sum of smaller

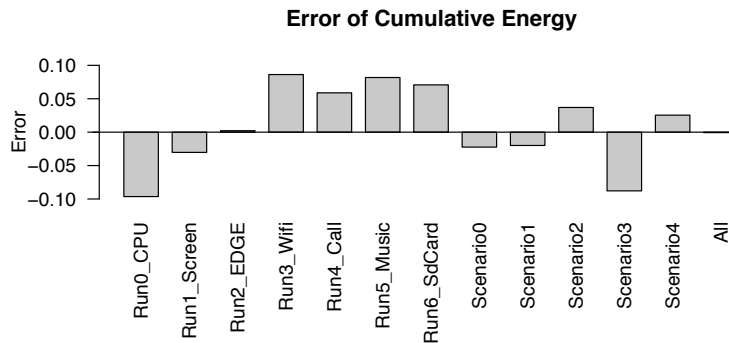


Figure 4.4. Cumulative total energy error.

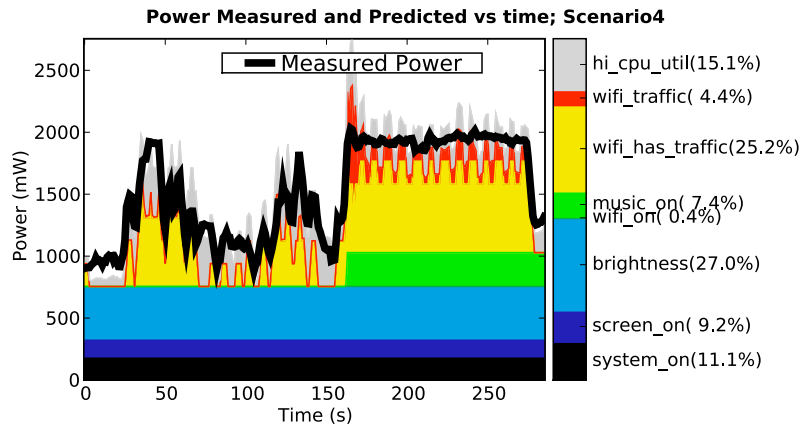


Figure 4.5. Power consumption timeline.

power components $p_{i,j}$ each corresponding to individual hardware units, the power contribution of an individual unit in the model may also be approximated, using Equation 4.1 in Section 4.2.2.

Figure 4.5 shows an example of the estimated and actual power over time for **Scenario4**. Each colored region represents the power $p_{i,j}$ attributed to a particular measurement, as labeled on the right of the plot. The total system power approximation at any point in

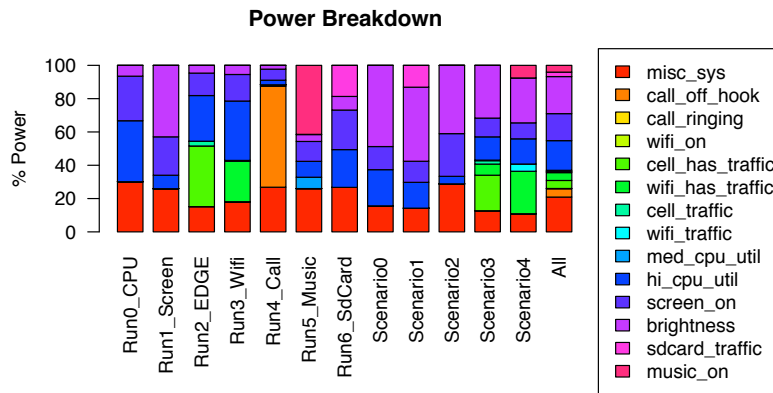


Figure 4.6. Power consumption breakdown for traces that stress specific hardware units.

time is represented by the top of the shaded regions. Measured system power is overlaid as a bold line. In this scenario, the user is surfing the Internet, then activates streaming media at 160 seconds. Our model estimates a total system power that closely tracks the actual measured power.

The same approach for per-component power approximation is applied to each of the logs used for the validation of our power estimation model. The estimated power from each product term in the linear regression model is accumulated and shown in Figure 4.6. The x-axis represents each of the logs, and each of the stacked bars corresponds to the power contribution from a specific parameter in our linear-regression-based power estimation model. The y-axis represents the percent of total power for the particular log. Test logs stressing particular components, such as EDGE or Wi-fi communications, have larger portions of power attributed to corresponding measurements.

The power breakdown for each of the test logs indicates that the relative power contribution per-component may vary drastically based upon the workload. For example,

during a phone call (shown in `Run4_Call`), over 50% of the total system power is consumed by the call. If only music is playing, and the screen is off (shown in `Run5_Music`), the DSP consumes significant power. The power breakdown is also dependent upon the system settings. For example, in `Scenario0`, the screen is at the highest brightness the entire time and dominates the power consumption of the system.

To better improve power consumption of any mobile platform, optimizations must target components with significant relative power consumption. However, as Figure 4.6 demonstrates, the per-component power breakdown widely varies with respect to the workload. Thus, it is important for architects to use representative workloads to characterize power consumption on mobile architectures. Such workloads should reflect the real user activity to correctly estimate the effect of any optimization.

Overall, our results show that (1) our high-level power estimation model can accurately predict the power consumption of the total system, (2) the power model can be used to derive a power breakdown of the total system, and (3) the power breakdown of a system is highly dependent upon the workload running on the mobile architecture.

4.3. Studying the User for Guiding Optimization

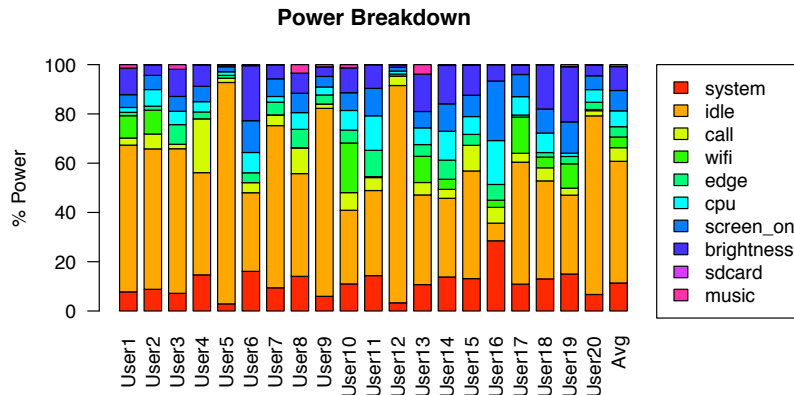
In this section, we explore the real user activity logs uploaded onto our server. We apply the power estimation model developed in Section 4.2 to characterize the power breakdown of mobile phones in the wild. We then present a study of active screen intervals, which suggest a potential power optimization for long screen intervals.

4.3.1. Characterizing Real User Workloads

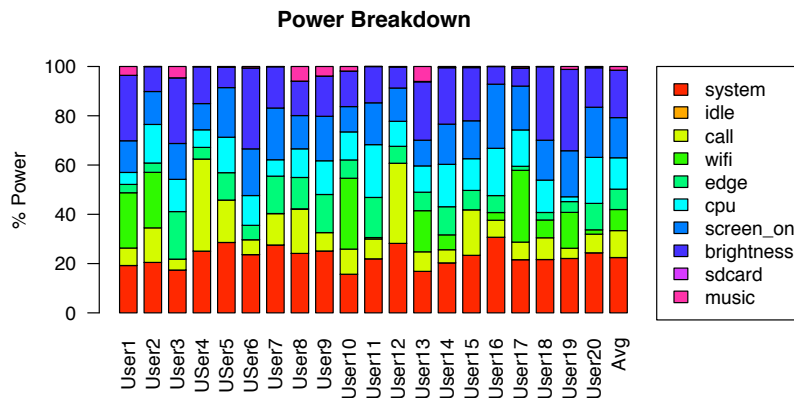
As described in Section 4.2.4 (and shown in Figure 4.6), the workload of a mobile architecture has a large effect on its power consumption; the hardware components that dominate power consumption vary drastically depending upon the workload. Since the user determines the workload for a mobile architecture, we must study real user behavior to understand the actual power consumption of mobile architectures in real environments. To this end, we have collected logs from users who have downloaded our logging application from Android Market, as described in Section 4.1. The logs contain the activity of 20 users, each for a duration exceeding a week, and accounting for approximately 250 days of user activity.

The power breakdown from each of the user logs is shown in Figure 4.7. Figure 4.7(a) shows the power breakdown including the estimated power contribution of the Idle state. To provide a clear breakdown of the power in the Active state, Figure 4.7(b) shows the same power breakdown excluding the samples from the Idle state. The x-axis represents each of the users. The product terms for each of the hardware components are combined for readability. The only exception is the screen, which is still shown separately as `screen_on` and `brightness`. We show both because the screen contributes heavily to the power breakdown of the system, and also because one of our optimizations (described later in Section 4.4) specifically targets reducing the screen brightness.

When examining the power consumption of the Idle state in Figure 4.7(a), two points are apparent. First, the power consumed during the Idle state can contribute to a significant fraction of the total system power consumption. The power consumed in the Idle state accounts for 49.3% of the total system power when averaging across all of the users.



(a) Power breakdown including idle time.



(b) Power breakdown excluding idle time.

Figure 4.7. Power consumption breakdown from real user traces.

Second, the fraction of total power consumed during the Idle state varies significantly across the users. At the extremes, the power consumption of Idle states contribute to 89.9% of the total power for User 5, but only 7.17% for User 16. This indicates that there is considerable variation in the usage patterns of mobile architectures across individual users.

When isolating the power consumption during the Active state (shown in Figure 4.7(b)), we again notice a large variation in the activity among all 20 users. For example, the power breakdown for User 4 and User 12 is dominated by the phone calls. User 6 and User 19 have their screen brightness set high, and thus, the brightness dominates their power breakdown. In addition, there is varying activity with regard to EDGE network usage versus Wi-fi network usage.

Overall, during Active usage time, two hardware components dominate the power consumption when averaging across all users: the screen and the CPU. The screen largely dominates the Active power breakdown and consumes 35.5% of the Active power; 19.2% due to the screen brightness and 16.3% due to the screen being on. The CPU accounts for 12.7% of the total Active power.

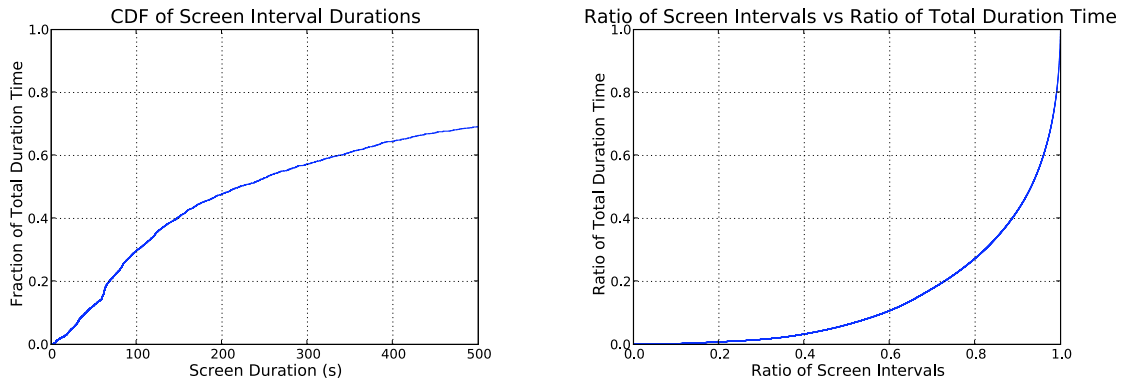
Although the Idle state may sometimes dominate the total system power, in this paper, we primarily focus on the power during the Active state. There are three reasons to be concerned with the Active state. First, the power consumed during the Idle state (≈ 68 mW) is significantly lower than the power that can be consumed in the Active state (up to 2000 mW when listening to music and using Wi-fi as shown in Figure 4.5). Second, the Active state contributes highly to the user experience since the user is actively engaged during the Active state. Any application that requires the apps processor would require the device to wake up and exit Idle mode. Finally, the Active state still accounts for a large fraction of the power consumed, accounting for 50.7% of the total system power.

4.3.2. Screen Usage of Real Users

Because the screen is the primary output device for interacting with the end user, the screen is a good indicator of user activity patterns. In addition, as we have shown in the previous section, it is the highest power consuming component on the device. We parse all of the user activity logs to extract *screen intervals*. A screen interval is a continuous block of time where the screen is on. The *duration* of a screen interval refers to the length of time that corresponds to the screen interval. The *total duration time* refers to the sum of durations for all screen intervals. From these intervals, we extract 9678 screen durations from our database, which accounts for 8.8 days worth of cumulative active “screen on” time.

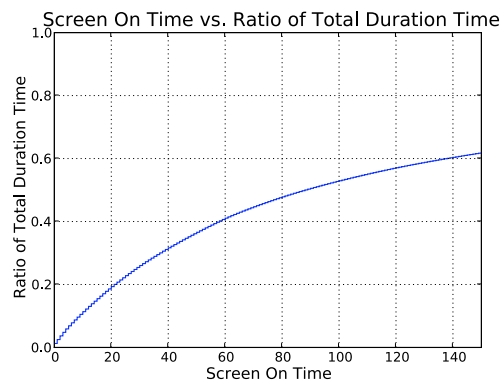
Figure 4.8(a) shows the cumulative fraction of the total duration for screen intervals up to 500 seconds. We see that screen intervals of 100 seconds or more constitute roughly 70% of the total screen duration (equivalently, as shown in Figure 4.8(a), 30% of the total screen duration is contributed by intervals shorter than 100 seconds). Figure 4.8(b) shows the cumulative distribution function (CDF) relating the total duration time to percentage of screen intervals. In other words, it shows the percentage of total duration time (shown on the y-axis) accounted for by the fraction of screen intervals with the shortest durations (x-axis). Figure 4.8(c) shows the ratio of *total* duration time accounted for by intervals shorter than a specific screen duration time, up to 150 seconds. This means that it shows the fraction of total duration if the first X seconds of each screen interval is considered.

Studying the screen intervals indicates that the total duration time is dominated by a relatively small percentage of long screen intervals. If we observe the 40 second mark on



(a) CDF of screen interval durations.

(b) Percentage of screen time relative to percentage of screen durations.



(c) Ratio of Total Screen Duration Constituted by Screen Interval

Figure 4.8. Screen durations based upon user activity.

the x-axis of Figure 4.8(b) and Figure 4.8(c), we see that the first 40 seconds of all screen intervals only accounts for 31% of the total duration time, but accounts for over 80% of all screen intervals. This means that if we have an optimization that saves power after 40 seconds of screen time, it would affect 69% of the total screen duration time, and only take effect in about 17% of the screen intervals. Based upon these observations, we conclude that it would be profitable to optimize for power during the long screen intervals. In the next section, we describe such an optimization.

4.4. User-Aware Optimization

As described in the Section 4.3, surprisingly, a few long screen intervals dominate the overall screen duration time. In addition, the power consumption during Active time is dominated by the screen and the CPU. To reduce the power consumption during these long intervals, we devise a scheme that reduces the brightness. Instead of simply dropping the brightness abruptly, we utilize *change blindness*, which is described in the next section. We also devise a similar scheme to control the CPU frequency.

4.4.1. Change Blindness

Researchers in human psychology and perception have revealed an inability for humans to detect large changes in their surrounding environment. One commonly-known study involves a video that prompts the viewer to count the number of times a basketball is passed. Halfway through the video, a man in a gorilla suit walks into the middle of the group of basketball players, thumps its chest, and then walks away. Surprisingly, the majority of viewers do not remember seeing a man in a gorilla suit, even though the concept is absurd and is in clear view in the middle of the video [96]. *Change blindness* refers to this inability for humans to detect large changes in their environment. The gorilla-suit study refers to change blindness of dynamic events, and occurs because although a human will view the entire video, their attention dictates the visual data that is processed. There have also been studies exploring change blindness in the case of gradual changes. Change blindness in the presence of gradual changes is more surprising as humans will miss significant changes without being distracted or disrupted. One study demonstrates change blindness as objects within images are removed from a picture, or

as the color of objects are slowly changed [97]. Another demonstrates change blindness as facial expressions are slowly changed in a picture.

We aim to utilize change blindness to reduce the power consumption of the device without causing any dissatisfaction to the users. Specifically, we devise schemes that reduce the screen brightness and CPU frequency slowly to save power. We compare these schemes to alternatives where the brightness and frequency are abruptly reduced and show that change blindness can indeed be utilized to save power consumption while minimizing the user dissatisfaction. We describe these schemes in the following section. To the best of our knowledge, this is the first study analyzing change blindness in the context of computer performance.

4.4.2. CPU Optimization

Existing DFS. The default system image used on the HTC Dream platform supports dynamic frequency scaling (DFS) on the ARM 11 apps processor, but uses a naïve DFS algorithm based upon the screen¹. If the apps processor is active and the screen is on, the processor is set to the highest frequency (384 MHz). If the apps processor is active and the screen is off, the processor is set to the middle frequency (246 MHz).

ondemand governor. A commonly used DFS scheme on desktop/server environments is the Linux `ondemand` DFS governor. The general algorithm is shown in Algorithm 3. At a high-level, the `ondemand` makes decisions based upon the CPU utilization. If the utilization is above a `UP_THRESHOLD`, it raises the CPU to the highest frequency. If the utilization is below a `DOWN_THRESHOLD`, it calculates the frequency that would maintain the

¹We have not found a confirmed description of this DFS scheme in any documentation on the HTC Dream, but have discovered this DFS behavior through our own experience with the device.

Algorithm 3: ondemand DFS algorithm.

```

1 procedure ONDEMAND-DFS(cpu_util)
2   if cpu_util  $\geq$  UP_THRESHOLD then
3     | SET-FREQUENCY(highest frequency)
4   else if cpu_util  $\leq$  DOWN_THRESHOLD then
5     | requested_freq  $\leftarrow$  frequency that maintains a utilization of at least
6     |   UP_THRESHOLD-10%
7     | SET-FREQUENCY(requested_freq)
8   return
9 procedure SET-FREQUENCY(freq)
10  if powersave_bias = 0.0 then
11    | Set CPU frequency to freq
12  else
13    | Alter CPU frequency dynamically to maintain an effective frequency of:
14    |   freq  $\times$  ((1000 - powersave_bias) * 0.001)
15  return
16

```

utilization below UP_THRESHOLD, and sets the frequency to that level. By setting the CPU frequency based upon CPU utilization, the `ondemand` governor saves power by reducing the frequency during times of low CPU utilization. We tune a knob within the `ondemand` governor called the `powersave_bias`, which is typically set to 0. The `powersave_bias` is a value between 0 and 1000 that specifies percentage with which to decrease the effective frequency of the CPU. `powersave_bias` increases in increments of 0.1%. A value of 0 indicates that the frequency should not be reduced at all. A value of 1000 indicates that the frequency should always be reduced by 100%, effectively reducing the CPU to its lowest frequency. If the `powersave_bias` indicates that the CPU frequency should be set to a frequency between two processor-supported frequencies, the `ondemand` governor will dynamically switch between the frequencies to simulate the frequency required.

Our DFS scheme: We use the `ondemand` governor and tune the `powersave_bias` knob leveraging change blindness for long screen intervals. Our DFS scheme hooks into

the screen events. Every four seconds, we increase the `powersave_bias` in increments of 30 (decrease effective frequency by 3%), until a maximum limit of 300 is reached. If the screen is turned off, the `powersave_bias` is reset back to 0. Thus, it reaches 70% of the frequency requested by `ondemand` within 40 seconds.

4.4.3. Screen Optimization

We implement a screen optimization to leverage change blindness that is similar to our CPU optimization. Again, we hook into the screen on and off events. We keep track of the user-set screen brightness. When the screen turns on, we set a timer for 3 seconds. Every 3 seconds, we decrease the brightness of the screen by 7 units (out of a maximum brightness of 255). We continue until the brightness reaches 60% of the user-set screen brightness and then stop. When the screen is turned off, we set the brightness back to the regular user-set screen brightness.

The idea in this scheme is to utilize two previous observations. First, since we slowly reduce the screen brightness, we will not reduce the power consumption on small screen intervals. However, as we have shown in the previous section, long screen intervals dominate the total screen duration, hence our optimization should still be able to save considerable fraction of the overall screen power consumption. Second, since our scheme reduces the screen brightness slowly, we expect that the users will be less likely to distinguish the change when compared to a sudden decrease in the screen brightness. Our experiments, described in Section 4.5, confirm that both of these goals are achieved.

4.5. Experimental Results

We now evaluate our optimizations described in Section 4.4. We refer to the two optimizations as *Screen Ramp* and *CPU Ramp*, for the screen and CPU optimizations, respectively. To test the change blindness hypothesis, we also introduce two more control schemes: *Screen Drop* and *CPU Drop*. Both of the *Drop* schemes wait 30 seconds before dropping to the respective minimum threshold levels of each of the change-blindness-inspired optimizations. In other words, the *Drop* and *Ramp* schemes eventually settle at the same brightness/frequency. However, the *Ramp* schemes slowly reach this destination, whereas the *Drop* schemes wait at the high brightness/frequency for the initial 30 seconds, before adjusting sharply to the final level.

We first evaluate the potential power savings of the optimization schemes by emulating the optimizations on the user logs. We then conduct a user study to assess user acceptance of our optimization schemes and to test our hypothesis on whether change blindness can be leveraged to optimize long screen intervals.

4.5.1. Power Savings

We approximate the power savings for each of our schemes by emulating the optimizations on user activity logs. To estimate the power savings of the screen optimizations, we adjust the brightness measurements in the logs to reflect the *Screen Drop* and *Screen Ramp* optimizations. Then, these new values are fed into the power model to generate the power consumption of the alternatives. To estimate the power savings of the CPU optimization, we first perform an estimation of the `ondemand` governor. If the CPU utilization is below 20%, we assume that `ondemand` would set the frequency to the lower

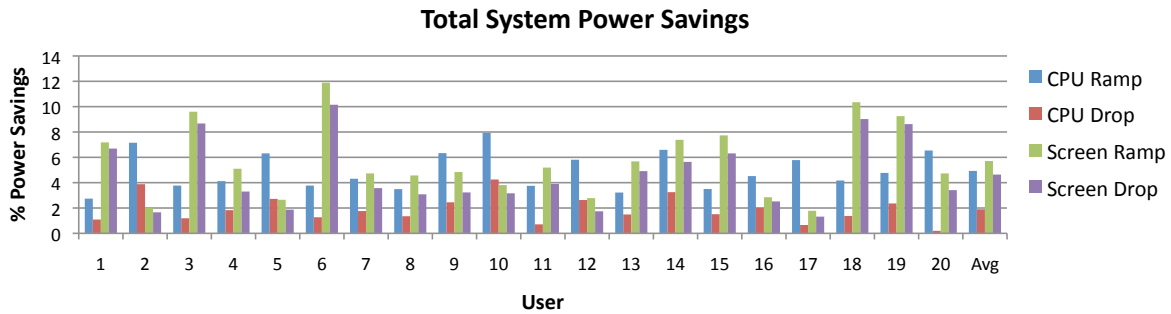


Figure 4.9. Total system power savings for each of the optimizations as estimated by our power model.

level. We then simulate our ramp-down mechanism on top of the `ondemand` scheme by multiplying the CPU product terms by a fraction that decreases in similar to *CPU Ramp* or *CPU Drop*. The newly-generated logs are processed with our power model to find the power consumption of *CPU Ramp*, *CPU Drop*, and `ondemand`.

Figure 4.9 shows the total system power savings when compared to the base scheme for each of the studied optimizations. On average, *CPU Ramp* saves 4.9% of the total system power. This corresponds to a 22.8% power savings when considering only the total CPU power. Of this, we estimate that 10.5% of the savings can be attributed to the base `ondemand` DFS governor, and the other 12.3% power savings is due to ramping the CPU frequency with the `powersave_bias`. The *CPU Drop*, on the other hand, achieves a total system power savings of 1.9%. When we compare the *CPU Ramp* to *CPU Drop*, we see that our *CPU Ramp* scheme achieves higher power savings. The reason for this result is the 30 second wait period of the *CPU Drop* scheme; while the *CPU Ramp* almost immediately starts reducing the CPU frequency, the *CPU Drop* scheme remains at the high frequency for 30 seconds, which causes a higher power consumption level. *Screen Ramp* saves 5.7% of total system power over all of the 20 users, and saves 19.1% of the

total screen brightness power. With *Screen Drop*, 4.6% of the total system power is conserved. Similar to the CPU schemes, when we compare the *Screen Ramp* to *Screen Drop*, we see that *Screen Ramp* achieves a higher power reduction level.

4.5.2. Impact on User Satisfaction

To evaluate the impact of our power saving techniques on the individual user satisfaction, we conduct another user study with 20 users. The user study involves three applications:

- **Web browsing:** Surfing Wikipedia with the web browser on the phone.
- **Game:** The BreakTheBricks game where the user moves a paddle on the bottom of the screen to bounce a ball and break a pattern of bricks.
- **Video:** The user watches a video with the PlayVideo application.

For each application, we perform six runs consisting of (1) *CPU Ramp*, (2) *CPU Drop*, (3) *Screen Ramp*, (4) *Screen Drop*, (5) *Ondemand*, and (6) the *Control*. The *Control* scheme is the default CPU and screen manager of the commercial phone. The order of runs are randomized so that the particular order is blind to the user as well as the proctor of the user study. After each run, we ask the user for a verbal user satisfaction rating ranging from 1 (Not Satisfied) to 5 (Satisfied).

Figure 4.10 shows the results of our user study for three applications. The three graphs show the user satisfaction ratings for each of the runs for the 20 users. Each of the graphs shows a set of clustered bars, each bar corresponding to the user satisfaction rating for a single run. At first glance, the average user satisfaction ratings look very similar for both Web Browsing and the Video, but they differ for the Game. For an in-depth analysis, we perform a paired t-test analysis for each application, comparing the set of

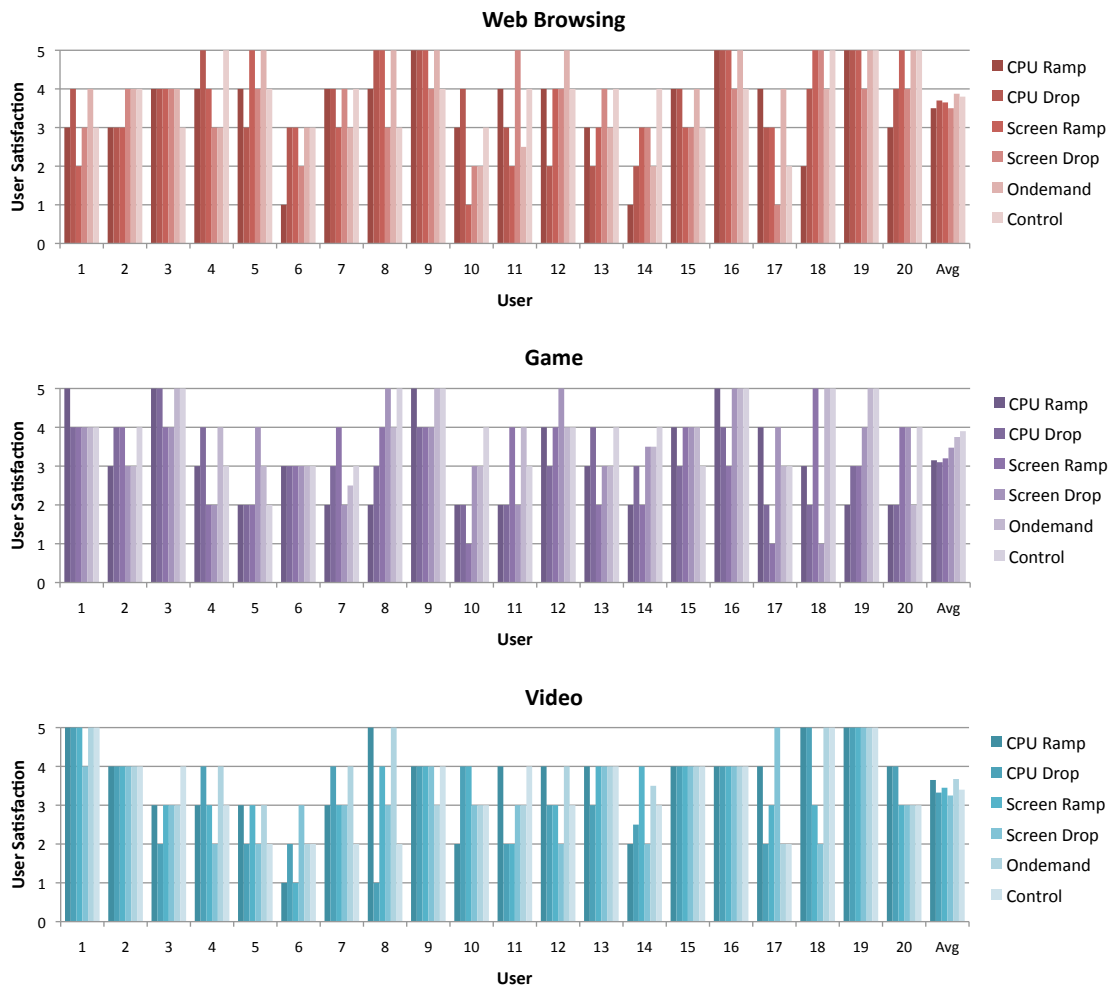


Figure 4.10. Reported user satisfaction for the (a) Web Browsing, (b) Game, and (c) Video applications.

user satisfaction ratings for each of the optimizations, to the set of user satisfaction ratings for the *Control* run. The paired t-test shows that there are five comparisons against the *Control* scheme where there is a statistically significant difference. In all other cases, there were no statistically significant changes between the *Control* and the studied schemes. Among the five cases that show difference, the four are in the Game application; all the four studied techniques exhibit reduced user satisfaction when compared the *Control*.

The fifth comparison that differs is the *Screen Drop* for Web Browsing. Overall, when comparing the different schemes, *Screen Drop* was statistically different from the *Control* run for the Game and Web applications, and the other schemes (barring the *Ondemand* scheme) differed from the *Control* run only on the Game application.

4.5.3. User Feedback and Acceptance

At the end the study, we debriefed each user by informing them of the purpose of the experiment. We discussed the power characterization study, that the CPU and screen tended to dominate the power consumption, and introduced our different power saving schemes to them. Afterwards, we questioned each user about their opinions on our various power saving schemes for the screen and CPU.

When compiling notes on the discussions with the users, we recognized two general trends:

- Most users determine their user satisfaction based upon how smoothly the computer responds to their input. 9 of the 20 users let us know they rated runs poorly when there were pauses, or the screen became jumpy/jittery. Our schemes performed the worst on the Game because any glitch would affect the smoothness of the bouncing ball and would be immediately noticeable. A result of this is that the rate of change for the CPU frequency does not matter – once the application becomes jittery, user satisfaction decreases. This trend is another reason why we did not observe a difference between the *CPU Ramp* and *CPU Drop* schemes; both of these schemes cause jitters after a certain CPU level is reached and regardless of how slowly we reduce the frequency, the glitches are noticeable.

Hence the users provided the same level of satisfaction for these two alternatives. However, we must note that the *CPU_Ramp* achieves a higher power saving when compared to *CPU_Drop*.

- Change blindness can be leveraged for the screen. 8 out of the 20 users noticed the drop in screen brightness during *Screen Drop* experiments. Only one of the users noticed the screen slowly dimming during *Screen Ramp*. In fact, almost all of the users were surprised when we told them that the screen was being slowly dimmed. As a result of this, we also observe that the users were less satisfied with the *Screen Drop* on the Web application, whereas they showed the same satisfaction level with the *Control* and *Screen Ramp* schemes.

At the end of the user study, we also asked the users whether they would turn a combination of these schemes on if they had a tool to control them and knew they would save about 10% of their battery life. Out of the 20 users, 15 said that they would use these optimizations, 1 was apathetic, and 4 of the users would not use the optimizations. Out of the 15 that responded with a yes, 5 of them expressed a desire for application-dependent optimization. For example, while they were fine with the dimming for the web application, they preferred a brighter screen for the video.

In summary, our results show that we can achieve significant reduction in power consumption by considering user activity patterns. Our *Screen_Ramp* and *CPU_Ramp* schemes reduce the power consumption by 5.7% and 4.9%, respectively, achieving a combined power saving of 10.6%. We also show successfully demonstrate power optimizations based upon indiscernible changes on the system parameters.

4.6. Summary

In this chapter, we have studied mobile architectures in their natural environment – in the hands of the end user. We present tools and methods for collecting and analyzing logs of real activity patterns for characterizing the power consumption and guiding optimization of mobile architectures accordingly. We build a logger application for the Android G1 phone and release it to the general public to collect logs from real users on real devices. We then develop a linear-regression-based power estimation model, which uses high-level measurements of each hardware component, to estimate the system power consumption. We show that the power estimation model is highly accurate and that it can provide insights about the power breakdown of the hardware components in a mobile architecture. By analyzing the user logs, we find that the power breakdown of a device is highly dependent upon the individual user, but that the screen and the CPU tend to dominate the active power consumption. We then demonstrate an example of leveraging user behavior to identify new optimizations. Specifically, we study active screen intervals and discover that a relatively small number of long screen intervals dominate the active screen time. Based upon this observation, we develop an optimization for the screen and the CPU that advantage of change blindness. We demonstrate that our optimizations can save up to 10% total system power while minimally impacting user satisfaction.

CHAPTER 5

Related Work

5.1. The Empathic Systems Project

The work in this dissertation lies within the larger context of the Empathic Systems Project at Northwestern University [40]. The project seeks to explore incorporating end-user satisfaction and guidance into the design and implementation of computer architectures and systems [33]. There are several related work from my colleagues in this research project.

Gupta [48], and Lin [65] demonstrate a high variation in user tolerance for resource borrowing. Mallik [74] and Lin [67] propose leveraging this user variation for CPU frequency tuning. Lins propose using direct user input to scheduling virtual machines [66] subject to individual user satisfaction. Miller shows that this variation can be leveraged for improvin the home network [77, 78]. We build this work and considers explicit and implicit methods of learning the relationship between hardware performance and user satisfaction for controlling CPU frequency.

PICSEL [73] controls CPU frequency based upon changes, and the rate of change, in pixels on the graphics display. The main idea is that for interactive GUI-based applications, users can only percieve changes that are visible via changes in the display. Thus, the quality of the graphics on the display may be a good proxy for perceivable performance.

Lange develops a speculative remote display [63] for predicting user actions to speculatively execute actions in a virtual network computing system. They leverage direct user input to manage the trade-off between responsiveness and screen correctness subject to the individual end user.

Tarzia proposes using sonar on computing systems to detect user attention and presence to improve currently power management policies [101, 102].

Shye, et al. build upon the mobile smartphone work in Chapter 4 to make characterize and model on real user activity on the Android G1 smartphone [95, 94].

5.2. Other User-Related Work

In addition, there are several related works that study the role of the end user in computer architecture and systems research. Maclean [71] and Sousa [57] both make cases for user-tailorable systems.

There has been work that takes user perception into account. These studies rely on high-level metrics, such as system response time. Endo et al. [38, 37] explore using latency as a performance metric and for detecting performance anomalies in operating systems. Olshefski studies latency of the network to infer client response time [80]. Vertigo [43] monitors application messages to measure user-perceived latency. Vertigo also proposes a layered frequency scaling scheme similar to PTP. Other DVFS algorithms use task information, such as measuring response times in interactive applications or rate of change in the display [68, 73] as a proxy for the user. However, they have used system-level metrics as a proxy for user satisfaction. This work directly correlates explicit and implicit measures of user satisfaction for making architecture-level decisions.

Zilles proposes increasing interactivity by predicting user actions [118]. Davison also studies the predictability of user actions [30].

Bi proposes IADVS [14], a DVFS scheme based upon predicting the CPU utilization following user input events. Vertigo [43] monitors application messages and can be used to perform the optimizations implemented in our study (although to the best of our knowledge this has not been studied). However, compared to Vertigo, our approach provides a metric/framework that is much easier to use.

Anand, Nightingale, and Flinn [6] discuss the concept of a control parameter that could be used by the user. However, they focus on the wireless networking domain, not the CPU. Second, they do not propose or evaluate a user interface.

Falaki studies real smartphone usage and find a wide variability in smartphone usage [41]. Phillips studies user activity for predicting when to sleep for wireless mobile devices [83]. MyExperience [44] gathers traces from user phones in the wild, similar to our work, but uses the traces to study high-level user actions. We study user activity patterns to understand system performance and for saving power on mobile architectures.

Outside of computer architecture and systems, the end user has been studied in a number of contexts. Some examples include incorporating the end user for improving internet security with CAPTCHAs [109], solving difficult AI problems via computer games [110, 108], modeling the user for improving video streaming [70], studying the perceptual quality of a media [28, 29, 61, 85, 90], and human-computer interaction researchers develop applications for improving the human condition [24, 25, 26].

5.3. Measuring the End User

The Affective Computing Group at MIT has worked to develop emotion-aware computers [84]. They have proposed devices such as HandWave GSR [99] with a squeezable mouse [87]. Their most related work is concerned with creating [89] or detecting [60] user frustration with learning software. There is also work on relating posture to persistence in puzzle games [5], and using face recognition software to improve social-emotional learning for autistic children [103]. Other researchers, such as Mandryk and Atkins [75] and Hazlett and Benedek [52], have also shown that physiological measures (e.g., GSR, EMG sensors, and heart rate) can be used to predict emotion when playing games. Our work, on the other hand, measures physiological responses in the face of changes in computer performance and utilize real-time sensing of physiological traits in making architectural decisions.

5.4. Power Modeling

Power modeling and estimation has been heavily studied from various angles. Wattch estimates microprocessor power consumption using low-level architectural features [20]. Several researchers use performance counters to estimate the power consumption of both high-performance and embedded microprocessors [13, 15, 27, 58, 59]. Gurun uses performance counters and communication measurements to estimate power consumption on an iPaq [50]. Cignetti uses power measurements to derive a power breakdown for Palm devices [23]. Tan uses function-level power models for software-implemented power estimation [100]. The power consumption of the operating system has been explored for high-performance [64] and embedded platforms [12, 32]. We differ from prior art by

developing a software-implemented, system-level power model that uses easily-accessible measurements and does not require specialized hardware (e.g., hardware performance counters) or software (e.g., hooks into the operating system).

SoftWatt uses simulation to understand the power consumption of the processor, memory, and disk on a high-performance architecture [49]. Mahesri measures the power breakdown on a laptop and discovers that the hardware components which dominate power consumption change depending upon the workload [72]. We use our power estimation model and traces to estimate the power breakdown of mobile architectures used by real users in real environments.

5.5. Dynamic Voltage and Frequency Scaling

Dynamic voltage and frequency scaling (DVFS) is an effective technique for microprocessor energy and power control for most modern processors [19, 45]. Energy efficiency has traditionally been a major concern for mobile computers. Fei, Zhong and Ya [42] propose an energy-aware dynamic software management framework that improves battery utilization for mobile computers. However, this technique is only applicable to highly-adaptive mobile applications. Researchers have proposed algorithms based on workload decomposition [22], but these tend to provide power improvements only for memory-bound applications. Wu et al. [114] present a design framework for a run-time DVFS optimizer in a general dynamic compilation system. The Razor [39] architecture dynamically finds the minimal reliable voltage level. Dhar, Maksimovic, and Kranzen [31] propose an adaptive voltage scaling technique that uses a closed-loop controller targeted towards standard-cell ASICs. Intel Foxtan technology [111] provides a mechanism for

select Intel Itanium 2 processors to adjust core frequency during operation to boost application performance. To the best of our knowledge, none of the previous DVFS techniques consider the user satisfaction prediction.

Sasaki et al. [88] propose a novel DVFS method based on statistical analysis of performance counters. However, their technique needs compiler support to insert code for performance prediction. Furthermore, their technique does not consider user satisfaction while setting the frequency. The primary contribution of our work is to establish the correlation between hardware counters and user satisfaction and utilize this correlation to develop a user-aware DVFS technique.

Other DVFS algorithms use task information, such as measured response times in interactive applications [68, 116] as a proxy for the user.

Xu, Ross, and Melhem propose novel schemes [115] minimizing energy consumption in real-time embedded systems that execute variable workloads. However, they try to adapt to the variability of the workload rather than to the users.

5.6. Screen Optimizations

Researchers have studied screen optimizations that would be enabled with OLED display technology. They explore altering the user interface to dim certain parts of screen to save considerable power, and do a user acceptance study [16, 51]. Our work operates on existing screen technology and leverages change blindness with gradual changes to save power.

CHAPTER 6

Conclusion

This dissertation makes the case for incorporating the end user into the design and optimization of modern computer architecture. It explores three aspects of human-computer interaction – user perception, user state, and user activity – and shows that studies of all three provide important insights to the design for designing and optimizing computer architectures and systems. With respect to user perception, the main take-away is that there is no average user. User studies show a significant variation across all users. This user variation represents a new opportunity for optimization architectures subject to individual user satisfaction. With respect to user physiological traits, I propose empathic inputs devices that exist solely to provide the computer with information about user state. I show that three empathic input devices (eyetrackers, a galvanic skin response sensor, and force sensors) can be used to infer user satisfaction during computer usage, and that this information can be leveraged to drive user-aware optimizations. With respect to user activity, I show that studying user activity can be critical to understanding the power consumption of mobile devices, as well as unveiling new properties of machine workload that can be leveraged for optimization.

At a higher level, the work in this dissertation points towards a new way of approaching decisions at the architecture and systems level. It suggests including the user at all levels of the design process. We should perform user studies, as well as studies of user activity, to find new opportunities for optimization. We should incorporate information we learn

directly from an individual user to drive optimizations. Currently, optimizations are driven by heuristic, or targeted at an average user. Instead, we should learn about the end user to tune machine performance to user expectations. Third, when evaluating design or optimization decisions, we must bring the user into the loop. The ultimate measure is whether the user is satisfied with performance or not, and this is impossible without real user feedback via user studies.

If we incorporate the end user into architecture and systems research, it implies a win-win scenario for both the computer and the user. The computer should execute more efficiently, performing tasks and tuning performance subject to the individual user. At the same time, we should substantially improve user satisfaction by providing a new user experience, with computer execution tailored to their individual needs. Both are good things.

References

- [1] Microsoft word 2000. Microsoft Corporation.
- [2] Tetris arena. Terminal Studio.
- [3] Need for speed prostreet, 2007. Electronic Arts.
- [4] Advanced Micro Devices. BIOS and Kernel Developer's Guide for AMD Athlon64 and AMD Opteron Processors. Technical report, Advanced Micro Devices, 2006.
- [5] H. I. Ahn, A. Teeters, A. Wang, C. Breazeal, and R. W. Picard. Stoop to conquer: Posture and affect interact to influence computer user's persistence. In *Proceedings of the Intl. Conference on Affective Computing and Intelligent Interaction*, September 2007.
- [6] M. Anand, E. B. Nightingale, and J. Flinn. Self-tuning wireless network power management. In *Proceedings of the Intl. Conference on Mobile Computing and Networking*, 2004.
- [7] J. Anderson, L. M. Berc, J. Dean, S. Ghemawat, M. R. Henzinger, S. Leung, R. L. Sites, M. T. Vandevoorde, C. A. Waldspurger, and W. E. Weihl. Continuous profiling: Where have all the cycles gone? In *Proc. of the 16th ACM Symposium of Operating Systems Principles*, pages 1–14, October 1997.
- [8] Apple Inc. iPhone OS Technology Overview: About iPhone OS Development, October 2008.
- [9] Arbitron and Edison Research Media. The Infinite Dial 2008: Radio's Digital Platforms.
- [10] A. Ax. The physiological differentiation between fear and anger in humans. *Psychosomatic Medicine*, 15(5):433–442, July 1952.
- [11] R. Azimi, M. Stumm, and R. W. Wisniewski. Online performance analysis by statistical sampling of microprocessor performance counters. In *Proceedings of the International Conference on Supercomputing*, June 2005.
- [12] K. Baynes, C. Collins, E. Fiterman, B. Ganesh, P. Kohout, C. Smit, T. Zhang, and B. Jacob. The performance and energy consumption of three embedded real-time operating systems. In *Proceedings of the Intl. Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, pages 203–210, November 2001.
- [13] F. Bellosa. The benefits of event-driven energy accounting in power-sensitive systems. In *Proceedings of the SIGOPS European Workshop*, September 2000.
- [14] M. Bi, I. Crk, and C. Gniady. Iadvs: On-demand performance for interactive applications. In *Proceedings of the Intl. Symposium on High-Performance Computer Architecture*, Jan 2010.
- [15] W. L. Bircher, M. Valluri, J. Law, and L. K. John. Runtime identification of microprocessor energy saving opportunities. In *Proceedings of the Intl. Symposium on Low Power Electronics and Design*, pages 275–280, 2005.

- [16] L. Bloom, R. Eardley, E. Geelhoed, M. Manahan, and P. Ranganathan. Investigating the relationship between battery life and user acceptance of dynamic, energy-aware interfaces on handhelds. In *Proceedings of the Intl. Conference on Human-Computer Interaction with Mobile Devices and Services*, pages 13–24, September 2004.
- [17] M. Bohme, A. Meyer, T. Martinetz, and E. Barth. Remote eye tracking: State of the art and directions for future development. In *Proceedings of the Conference on Communication by Gaze Interaction*, pages 12–17, 2006.
- [18] W. Boucsein. *Electrodermal Activity*. Plenum Press, 1992.
- [19] B. Brock and K. Rajamani. Dynamic power management for embedded systems. In *Proceedings of IEEE SOC Conference*, 2003.
- [20] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *Proceedings of the Intl. Symposium on Computer Architecture*, pages 83–94, 2000.
- [21] S. Browne, J. Dongarra, N. Garner, G. Ho, and P. Mucci. A portable programming interface for performance evaluation on modern processors. *The International Journal of High Performance Computing Applications*, 14(3):189–204, 2000.
- [22] K. Choi, R. Soma, and M. Pedram. Dynamic voltage and frequency scaling based on workload decomposition. In *Proceedings of the Intl. Symp. on Low Power Electronics and Design*, 2004.
- [23] T. L. Cignetti, K. Komarov, and C. S. Ellis. Energy estimation tools for the PalmTM. In *Proceedings of the Intl. Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, August 2000.
- [24] S. Consolvo, P. Klasnja, D. W. McDonald, and J. A. Landay. Goal-setting considerations for persuasive technologies that encourage physical activity. In *Proceedings of the International Conference on Persuasive Technology*, 2009.
- [25] S. Consolvo, K. Markle, K. Patrick, and K. Chanasyk. Designing for persuasion: mobile services for health behavior change. In *Proceedings of the Conference on Persuasive Technology*, 2009.
- [26] S. Consolvo, D. W. McDonald, and J. A. Landay. Theory-driven design strategies for technologies that support behavior change in everyday life. In *Proceedings of the International Conference on Human Factors in Computing Systems*, pages 405–414, 2009.
- [27] G. Contreras and M. Martonosi. Power Prediction for Intel XScale[®] Processors Using Performance Monitoring Unit Events. In *Proceedings of the Intl. Symposium on Low Power Electronics and Design*, pages 221–226, August 2005.
- [28] N. Cranley, L. Murphy, and P. Perry. User-perceived quality-aware adaptive delivery of mpeg-4 content. In *Proceedings of the international workshop on Network and operating systems support for digital audio and video*, pages 42–49, 2003.
- [29] N. Cranley, P. Perry, and L. Murphy. User perception of adapting video quality. *Int. J. Hum.-Comput. Stud.*, 64(8):637–647, 2006.
- [30] B. D. Davison and H. Hirsh. Predicting sequences of user actions. In *Proceedings of the Workshop on Predicting the Future*, 1998.
- [31] S. Dhar, D. Maksimovic, and B. Kranzen. Closed loop adaptive voltage scaling controller for standard cell asics. In *Proceedings of Intl. Symp. on Low Power Electronics and Design*, 2005.

- [32] R. P. Dick, G. Lakshminarayana, A. Raghunathan, and N. K. Jha. Power analysis of embedded operating systems. In *Design Automation Conference*, pages 312–315, 2000.
- [33] P. Dinda, G. Memik, R. Dick, B. Lin, A. Mallik, A. Gupta, and S. Rossoff. The user in experimental computer systems research. In *Proceedings of the Workshop on Experimental Computer Science*, June 2007.
- [34] Display Search; NPD Group. Strong Mini-Note PC Demand Expected to Buoy Notebook Market in 2009, April 2009. <http://www.displaysearch.com/>.
- [35] N. Eagle and A. Pentland. Social serendipity: Mobilizing social software. *IEEE Pervasive Computing*, 4(2):28–34, January–March 2005.
- [36] W. Einhauser, J. Stout, C. Kock, and O. Carter. Pupil dilation reflects perceptual selection and predicts subsequent stability in perceptual rivalry. In *Proceedings of the National Academy of Sciences*, pages 1704–1709, 2008.
- [37] Y. Endo and M. I. Seltzer. Improving interactive performance using tipme. In *Proceedings of the Intl. Conference on Measurements and Modeling of Computer Systems*, 2000.
- [38] Y. Endo, Z. Wang, J. B. Chen, and M. I. Seltzer. Using latency to evaluate interactive system performance. In *Proceedings of the USENIX Symp. on Operating Systems Design and Implementation*, 1996.
- [39] D. Ernst, N. S. Kim, S. Das, S. Pant, T. Pham, R. Rao, C. Ziesler, D. Blaauw, T. Austin, and T. Mudge. Razor: A low-power pipeline based on circuit-level timing speculation. In *Proceedings of the Intl. Symposium on Microarchitecture*, 2003.
- [40] ESP: Empathic Systems Project. <http://www.empathicsystems.org>.
- [41] H. Falaki, R. Mahajan, S. Kandula, D. Lymberopoulos, R. Govindan, and D. Estrin. Diversity in smartphone usage. In *Proceedings of Intl. Conf. on Mobile Systems, Applications and Services*, June 2010.
- [42] Y. Fei, L. Zhong, and N. K. Jha. An energy-aware framework for coordinated dynamic software management in mobile computers. In *Proceedings of Intl. Symp. on Modeling, Analysis and Simulation of Computer and Telecommunications Systems*, 2004.
- [43] K. Flautner and T. N. Mudge. Vertigo: Automatic performance setting for linux. In *Proceedings of the Symposium on Operating Systems Design and Implementation*, 2002.
- [44] J. Froehlich, M. Y. Chen, S. Consolvo, B. Harrison, and J. A. Landay. MyExperience: A system for in situ tracing and capturing of user feedback on mobile phones. In *MOBISYS*, 2007.
- [45] S. Gochman and R. Ronen. The Intel Pentium M processor: Microarchitecture and performance. *Intel Technology Journal*, 2003.
- [46] Google, Inc. Android - An Open Handset Alliance Project. <http://developer.android.com>.
- [47] Google, Inc. Corporate Information - Our Philosophy. <http://www.google.com/corporate/tenthings.html>.
- [48] A. Gupta, B. Lin, and P. A. Dinda. Understanding user comfort with resource borrowing. In *Proceedings of the Intl. Symp. on High Performance Distributed Computing*, 2004.
- [49] S. Gurumurthi, A. Sivasubramaniam, M. J. Irwin, N. Vijaykrishnan, M. Kandemir, T. Li, and L. K. John. Using Complete Machine Simulation for Software Power Estimation: The SoftWatt Approach. In *Proceedings of the Intl. Symposium on High Performance Computer Architecture*, pages 141–150, February 2002.

- [50] S. Gurun and C. Krintz. A run-time feedback-based energy estimation model for embedded devices. In *Proceedings of the Intl. Conference on Hardware/Software Codesign and System Synthesis*, pages 28–33, October 2006.
- [51] T. Harter, S. Vroegindewij, E. Geelhoed, M. Manahan, and P. Ranganathan. Energy-aware user interfaces: An evaluation of user acceptance. In *Proceedings of the Conference on Human Factors in Computing Systems*, pages 199–206, April 2004.
- [52] R. L. Hazlett and J. Benedek. Measuring emotional valence to understand the user’s experience of software. *International Journal of Human-Computer Studies*, 65:306–314, 2007.
- [53] Hewlett-Packard Development Company. perfmon project <http://www.hpl.hp.com/research/linux/perfmon/>.
- [54] Intel Corporation. *Intel 64 and IA-32 Architecture Software Developer’s Manual Volume 3A: System Programming Guide*. Santa Clara, CA, 2002.
- [55] Intel Corporation. Intel Itanium 2 processor reference manual: For software development and optimization. May 2004.
- [56] S. T. Iqbal, P. D. Adamczyk, Z. S. Zheng, and B. P. Bailey. Towards an index of opportunity: Understanding changes in mental workload during task execution. In *Proceedings of the Conference on Human Factors in Computing Systems (CHI)*, pages 311–320, April 2005.
- [57] Joao P. Sousa and Rajesh K. Balan and Vahe Poladian and David Garlan and Mahadev Satyanarayanan. Giving users the steering wheel for guiding resource-adaptive systems. Technical Report CMU-CS-05-198, Carnegie Mellon University, School of Computer Science, Dec 2005.
- [58] R. Joseph and M. Martonosi. Run-time power estimation in high performance microprocessors. In *Proceedings of the Intl. Symposium on Low Power Electronics and Design*, August 2001.
- [59] I. Kadayif, T. Chinoda, M. T. Kandemir, N. Vijaykrishnan, M. J. Irwin, and A. Sivasubramaniam. vec: virtual energy counters. In *Proceedings of the Workshop on Program Analysis For Software Tools and Engineering*, June 2001.
- [60] A. Kapoor, W. Burleson, and R. W. Picard. Automatic prediction of frustration. *Intl. Journal of Human-Computer Studies*, pages 724–736, August 2007.
- [61] J.-G. Kim, Y. Wang, and S.-F. Chang. Content-adaptive utility-based video adaptation. In *Proceedings of the International Conference on Multimedia and Expo*, pages 281–284, 2003.
- [62] J.-O. Klein, J.-O. Klein, L. Lacassagne, H. Mathias, S. Moutault, and A. Dupret. Low power image processing: Analog versus digital comparison. In *CAMP ’05: Proceedings of the Seventh International Workshop on Computer Architecture for Machine Perception*, pages 111–115, Washington, DC, USA, 2005. IEEE Computer Society.
- [63] J. Lange, P. A. Dinda, and S. Rossoff. Experiences with client-based speculative remote display. In *Proceedings of the USENIX Annual Technical Conference*, June 2008.
- [64] T. Li and L. K. John. Run-time modeling and estimation of operating system power consumption. In *SIGMETRICS*, 2003.
- [65] B. Lin and P. A. Dinda. Putting the user in direct control of cpu scheduling. In *Proceedings of the International Symposium on High Performance Distributed Computing*, June 2006.

- [66] B. Lin and P. A. Dinda. Towards scheduling virtual machines based on direct user input. In *Proceedings of the 1st International Workshop on Virtualization Technology in Distributed Computing*, Nov 2006.
- [67] B. Lin, A. Mallik, P. A. Dinda, G. Memik, and R. P. Dick. User- and process-driven dynamic voltage and frequency scaling. In *Proceedings of the International Symposium on Performance Analysis of Systems and Software*, April 2009.
- [68] J. Lorch and A. Smith. Using user interface event information in dynamic voltage scaling algorithms. Technical Report UCB/CSD-02-1190, University of California at Berkeley, Berkeley, CA, 2002.
- [69] J. Lu, H. Chen, P.-C. Yew, and W.-C. Hsu. Design and implementation of a lightweight dynamic optimization system. In *Journal of Instruction-Level Parallelism 6(2004)*, pages 1–24, April 2004.
- [70] C. E. Luna, L. P. Kandi, and A. K. Katsaggelos. Maximizing user utility in video streaming applications. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(2):141–148, February 2003.
- [71] A. MacLean et al. User-tailorable systems: Pressing the issues with buttons. In *Proceedings of the Conf. on Human factors in Computing Systems*, pages 175–182, April 1990.
- [72] A. Mahesri and V. Vardhan. Power consumption breakdown on a modern laptop, workshop on power aware computing systems. In *Proceedings of the Workshop on Power-Aware Computer Systems*, December 2004.
- [73] A. Mallik, J. Cosgrove, R. Dick, G. Memik, and P. Dinda. PICSEL: Measuring user-percieved performance to control dynamic frequency scaling. In *Proceedings of the Intl. Conference on Architectural Support for Programming Languages and Operating Systems*, March 2008.
- [74] A. Mallik, B. Lin, G. Memik, P. A. Dinda, and R. P. Dick. User-driven frequency scaling. *Computer Architecture Letters*, 5(2), July–December 2006.
- [75] R. L. Mandryk and M. S. Atkins. A fuzzy physiological approach for continuously modeling emotion during interaction with play technologies. *International Journal of Human-Computer Studies*, 65:329–347, 2007.
- [76] Microsoft. Windows native processor performance control. In *Windows Platform Design Notes*, November 2002.
- [77] J. S. Miller, J. R. Lange, and P. A. Dinda. EmNet - Satisfying the Individual End User Through Empathic Home Networks. In *In proceedings. of the Intl. Conf. on Computer Communications*, March 2010.
- [78] J. S. Miller, A. Mondal, R. Potharaju, P. A. Dinda, and A. Kuzmanovic. Network Monitoring is People: Understanding End-User Perception of Network Problems. Technical report, Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL.
- [79] V. G. Moshnyaga and E. Morikawa. Reducing energy consumption of computer display by camera-based user monitoring. In *Lecture Notes in Computer Science*, pages 528–539, 2005.
- [80] D. P. Olshefski, J. Nieh, and D. Agrawal. Inferring client response time at the web server. In *Proceedings of the Intl. Conference on Measurement and Modeling of Computer Systems*, June 2002.

- [81] V. Pallipadi and A. Starikovskiy. The ondemand governor: Past, present, and future. In *Ottawa Linux Symposium*, July 2006.
- [82] T. Partala and V. Surakka. Pupil size variation as an indication of affective processing. *Int. J. Human-Computer Studies*, 59:185–198, 2003.
- [83] C. Phillips, S. Singh, D. Sicker, and D. Grunwald. Applying models of user activity for dynamic power management in wireless devices. In *Mobile HCI*, September 2008.
- [84] R. W. Picard. *Affective Computing*. MIT Press, Cambridge, 1997.
- [85] M. Pinson and S. Wolf. Comparing subjective video quality testing methodologies. In *SPIE Video Communications and Image Processing Conference*, pages 8–11, 2003.
- [86] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2009. ISBN 3-900051-07-0.
- [87] C. J. Reynolds. The sensing and measurement of frustration with computers. Master’s thesis, Master of Science in Media Arts and Technology at the MIT, Cambridge, MA, 2001.
- [88] H. Sasaki, Y. Ikeda, M. Kondo, and H. Nakamura. An intra-task DVFS technique based on statistical analysis of hardware events. In *Proceedings of the Intl. Conf. on Computing Frontiers*, 2007.
- [89] J. Scheierer, R. Fernandez, J. Klein, and R. W. Picard. Frustrating the user on purpose: A step toward building an affective computer. *Interacting with Computers*, 14(2):93–118, 2002.
- [90] K. Seshadrinathan, R. Soundararajan, A. C. Bovik, and L. K. Cormack. Study of subjective and objective quality assessment of video. *Transactions on Image Processing*, 19(6):1427–1441, June 2010.
- [91] A. Shye, B. Ozisikyilmaz, A. Mallik, G. Memik, P. A. Dinda, R. P. Dick, and A. N. Choudhary. Learning and leveraging the relationship between architecture-level measurements and individual user satisfaction. In *Proceedings of the Intl. Symposium on Computer Architecture*, June 2008.
- [92] A. Shye, Y. Pan, B. Scholbrock, J. S. Miller, G. Memik, P. A. Dinda, and R. P. Dick. Power to the people: Leveraging human physiological traits to control microprocessor frequency. In *Proceedings of the Intl. Symposium on Microarchitecture*, December 2008.
- [93] A. Shye, B. Scholbrock, and G. Memik. Into the wild: Studying real user activity patterns to guide power optimizations for mobile architectures. In *Proceedings of the Intl. Symposium on Microarchitecture*, December 2009.
- [94] A. Shye, B. Scholbrock, G. Memik, and P. A. Dinda. Characterizing and modeling user activity on smartphones. Technical Report NWU-EECS-10-06, Northwestern University, Evanston, IL, March 2010.
- [95] A. Shye, B. Scholbrock, G. Memik, and P. A. Dinda. Characterizing and modeling user activity on smartphones: Summary. In *Proceedings of the Intl. Conf. on Measurement and Modeling of Computer Systems (SIGMETRICS)*, June 2010.
- [96] D. J. Simons and C. F. Chabris. Gorillas in our midst: sustained inattention blindness for dynamic events. *Perception*, 28:1059–1074, 1999.
- [97] D. J. Simons, S. L. Franconeri, and R. L. Reimer. Change blindness in the absence of a visual disruption. *Perception*, 29:1143–1154, 2000.

- [98] T. J. Smith, M. Whitwell, and J. Lee. Eye movements and pupil dilation during event perception. In *Proceedings of the Eye Tracking Research and Applications Conference*, March 2006.
- [99] M. Strauss, C. Reynolds, S. Huges, K. Park, G. McDarby, and R. W. Picard. The hand-wave bluetooth skin conductance sensor. In *Proceedings of the Intl. Conference on Affective Computing and Intelligent Interaction*, October 2005.
- [100] T. K. Tan, A. Raghunathan, G. Lakshiminarayana, and N. K. Jha. High-level software energy macro-modeling. In *Proceedings of Design Automation Conference*, pages 605–610, June 2001.
- [101] S. Tarzia, R. P. Dick, P. A. Dinda, and G. Memik. Sonar-based measurement of user presence and attention. In *Proceedings of Intl. Conf. on Ubiquitous Computing*, September 2009.
- [102] S. Tarzia, R. P. Dick, P. A. Dinda, and G. Memik. Display power management policies and practice. In *Proceedings of Intl. Conf. on Autonomic Computing and Communications*, June 2010.
- [103] A. Teeters. User of a wearable camera system in conversation: Towards a companion tool for social-emotional learning in autism. Master’s thesis, Master of Science in Media Arts and Technology at the MIT, Cambridge, MA, 2001.
- [104] Tekscan. Flexiforce: System and sensor pricing. <http://www.tekscan.com/flexiforce/pricing.html>.
- [105] M. Toyokura. Waveform and habituation of sympathetic skin response. *Electroencephalography and Clinical Neurophysiology/Electromyography and Motor Control*, 109(2):178–183, 1998.
- [106] G. A. Tsihrintzis, M. Virvou, E. Alepis, and I. Stathopoulou. Towards improving visual-facial emotion recognition through use of complementary keyboard-stroke pattern information. In *Proceedings of the Intl. Conference on Information Technology: New Generations*, pages 32–37, April 2008.
- [107] R. Vetrugno, R. Liguori, P. Cortelli, and P. Montagna. Sympathetic skin response: Basic mechanisms and clinical applications. *Clinical Autonomic Research*, pages 256–270, June 2003.
- [108] L. von Ahn. Games with a purpose. *IEEE Computer*, 39(6):92–94, 2006.
- [109] L. von Ahn, M. Blum, N. Hopper, and J. Langford. Captcha: Using hard ai problems for security. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 294–311, May 2003.
- [110] L. von Ahn and L. Dabbish. Labeling images with a computer game. In *Proceedings of the Conf. on Human factors in Computing Systems*, pages 319–326, April 2004.
- [111] J. Wei. Foxton technology pushes processor frequency, application performance.
- [112] M. Whang. The emotional computer adaptive to human emotion. *Phillips Research: Probing Experience*, 8:209–219, 2008.
- [113] Wikipedia: The Free Encyclopedia. HTC Dream. <http://en.wikipedia.org/wiki/Gphone>.
- [114] Q. Wu, V. Reddi, Y. Wu, J. Lee, D. Connors, D. Brooks, M. Martonosi, and D. W. Clark. A dynamic compilation framework for controlling microprocessor energy and performance. In *Proceedings of the Intl. Symposium on Microarchitecture*, November 2005.

- [115] R. Xu, D. Moss, and R. Melhem. Minimizing expected energy in real-time embedded systems. In *Proceedings of the Intl. Conf. on Embedded Software*, 2005.
- [116] L. Yan, L. Zhong, and N. K. Jha. User-perceived latency based dynamic voltage scaling for interactive applications. In *Proceedings of the Design Automation Conference*, 2005.
- [117] D. Yang, A. Gamal, B. Fowler, and H. Tian. A 640×512 CMOS image sensor with ultrawide dynamic range floating-point pixel-level ADC. *Solid-State Circuits, IEEE Journal of*, 34(12):1821–1834, 1999.
- [118] C. Zilles. Increasing interactivity by predicting user actions. In *Wild and Crazy Ideas Session at the Intl. Conf. on Architectural Support for Programming Languages and Operating Systems*, 2004.

Vita

Dr. Alex Shye is an experimental computer architecture and systems researcher. His research involves prototyping real systems that improve the energy-efficiency, performance, and reliability of computer architectures. His dissertation explores incorporating the end user into the architectural design process by leveraging user perception, user physiological traits, and user activity. His other past research projects involve dynamic program profiling techniques, dynamic program compilation/optimization, dynamic memory allocation, and software-implemented transient fault tolerance.

Dr. Shye received his BS degree in Computer Engineering from the University of Illinois (2002), MS degree in Computer Engineering from the University of Colorado (2005), and PhD degree in Electrical and Computer Engineering from Northwestern University (2010).