Energy-Efficient Mobile System Design: The User's Perspective

Lin Zhong

A Dissertation

Presented to the Faculty

of Princeton University

in Candidacy for the Degree

of Doctor of Philosophy

Recommended for Acceptance

by the Department of

Electrical Engineering

September, 2005

Abstract

Mobile systems, such as smart-phones and Pocket PCs, are likely to become personal servers for future pervasive computing. This dissertation highlights the energy efficiency bottleneck such systems face in achieving this goal: the slow-user problem, i.e., an increasingly powerful computer spends most of its energy waiting for a constantly slow user. To tackle this bottleneck, it proposes to evaluate and improve the energy efficiency of a mobile system from the user's perspective.

The dissertation first highlights the slow-user problem and the importance of user interfaces and human factors. It offers a theoretical analysis for interface power and energy requirements, and a comprehensive energy characterization of state-of-the-art user interfaces. Based on this characterization, it presents a comparative study of their energy efficiency. It then tackles the slow-user problem in three different ways. First, it proposes to increase user productivity without much power overhead with a Bluetooth-based personal-area network of low-power interfacing devices. The network consists of a wrist-watch, single-hand single-tap multi-finger keypad, smart speech portal, and information-capturing devices. They synergistically provide a user with natural and power-efficient access to a mobile system. Second, it proposes to reduce the interfacing power requirement without decreasing user productivity much. Motivated by the memory-cache theory, the dissertation shows how a low-power interface cache device can be used to host simple interactive tasks outsourced from a mobile system, and thus improve the overall energy efficiency. Third, the dissertation proposes to aggressively power-manage a mobile system in its idle periods during human-computer interaction. It uses user interface information to predict user delays based on history and theories from the field of psychology. It shows that such a delay prediction can be combined with power management to significantly reduce idle power consumption.

Acknowledgments

It seems that most, if not all, dissertations start by thanking the advisor. It becomes hard to tell whether the advisor is special. Hence, my apology first to my advisor, Prof. Niraj K. Jha, for abusing his perfectionism in publications, including this dissertation. Advising twelve to sixteen Ph.D students during my five-year tenure in his group, he has never failed to return my draft with every typo and grammatical mistake corrected, in time. However, I always manage to skip some of his corrections. Therefore, all remaining typos and mistakes are mine alone. Prof. Jha has advised me on all the work included in this dissertation and all the electronic design automation (EDA) work I have done at Princeton. His accessibility, deep and broad knowledge have been a great fortune for not only me but the whole group. I have especially benefited from his unwillingness to solve the same problem more than once and his courage to venture into new fields. He will continue to be my model of a great researcher and mentor.

During my pursuit of Ph.D., I have been fortunate to have had help from or work with many people. Profs. Ruby B. Lee and Margaret R. Martonosi served on my Ph.D. oral examination committee. Prof. Ron Weiss encouraged me to look beyond a course project to explore the possibility of using cells for molecular electronics fabrication. I had the fortune to work with Drs. Anand Raghunathan and Srivaths Ravi as an intern at NEC Labs, America, in the summer of 2003 and Mr. Michael J. Sinclair as an intern at Microsoft Research in the summers of 2004 and 2005. I learned a great deal from my internship supervisors and am deeply indebted to them for their generous help to my career. Michael also worked with me very closely on the work in Chapter 6 performed during my first internship at Microsoft Research. I was also fortunate enough to collaborate with many members of Prof. Jha's group. Drs. Jiong Luo, Tat-Kee Tan, Weidong Wang, Li Shang,

Yunsi Fei, Keith Vallerio and I teamed up to develop a low-power high-level synthesis tool. Tat-Kee and Weidong developed the power measurement toolkit used in Chapters 4 and 8. Yunsi and I worked on dynamic software management on mobile systems. Keith and I extended the work in Chapter 4 to energy-efficient graphical user interface design. Pallav Gupta and I worked on interconnect power modeling. We later on worked with Rui Zhang on EDA for nanotechnologies. Le Yan and I collaborated on making operating systems more responsive to users without sacrificing power efficiency. I thank all my collaborators for the opportunity to work with them and all the things that I learned from them. All works included in this dissertation have appeared in refereed publications. Blind reviewers helped us refine them before they appeared in publication, and helped us broaden our thinking. I thank them all for their criticism and suggestions. I am honored to have Prof. Niraj K. Jha, Dr. Anand Raghunathan, and Dr. Srivaths Ravi as my dissertation readers whose comments have helped make this dissertation much better. Pallav and Chao Huang helped me with many administrative issues while I was finishing this dissertation away from Princeton.

Many people offered a lot of help during my academic job search. I was honored to have Prof. Niraj K. Jha, Prof. Ron Weiss, Prof. Li-Shiuan Peh, Dr. Anand Raghunathan, and Mr. Michael J. Sinclair as my references. Stacey Weber and Sarah Braude provided significant assistance. Former Princeton students, Drs. Robert Dick, Yuan Xie, Zhijie Shi, and Yunsi Fei offered me invaluable counseling.

I would also like to thank my many colleagues and friends in the Princeton and Seattle areas, where great things always happen.

Finally, I thank Yuanyuan for making everything meaningful and for helping me become a better person everyday.

Contents

Abstract **iii**

Acknowledgments **iv**

Contents **vi**

List of Figures **x**

List of Tables **xii**

1 Introduction **1**

1.1 Mobile systems: The future of personal computing 1

1.1.1 The convergence of evolution 2

1.1.2 The blessing of computing and curse of interfaces 3

1.2 Energy efficiency and the slow-user problem 5

1.3 Energy efficiency: The user's perspective 7

1.4 Dissertation contributions 8

2 Related Work **10**

2.1 System view of a mobile system 10

2.2 Energy optimization 12

2.2.1 Hardware activity reduction 12

2.2.2 Hardware activity balance 13

2.2.3 Power management 14
2.2.4 Performance scaling 19
2.3 System support for energy optimization 26
2.3.1 Compilation for performance scaling 26
2.3.2 OS support 27
2.4 Energy analysis and characterization 30
2.5 User interfaces and human factors 31
2.6 Chapter summary 32

3 Energy Efficiency Limits Imposed by Human Factors 33
3.1 Sensory perception-based limits 33
3.1.1 Visual output 34
3.1.2 Auditory output 35
3.1.3 Power reduction techniques 36
3.2 Input/output speed 37
3.3 Chapter summary 38

4 Energy Consumption of Graphical User Interfaces 39
4.1 Background 39
4.1.1 Mobile software 40
4.1.2 Mobile GUI platforms 40
4.2 GUI energy consumption 42
4.3 Experimental setup 45
4.3.1 System information 45
4.3.2 Energy measurement 46
4.3.3 Methodology and benchmarks 46
4.4 GUI energy characterization 50
4.5 GUI design for energy efficiency 59
4.6 Chapter summary 61

5 Energy Efficiency of Mobile User Interfaces 62

5.1 Characterization setup . . . 62

5.2 Visual interfaces . . . 64

5.3 Auditory interfaces . . . 65

5.3.1 Direct recording and playback . . . 66

5.3.2 Speech recognition and synthesis . . . 66

5.4 Manual input techniques . . . 68

5.5 A comparative study . . . 69

5.5.1 Output . . . 70

5.5.2 Input . . . 71

5.6 Observations . . . 75

5.7 Chapter summary . . . 77

6 Pervasive Interfacing: A Personal-Area Network of Wireless Interfacing Devices 78

6.1 Design principles . . . 79

6.2 Bluetooth-based PAN . . . 80

6.2.1 Bluetooth . . . 81

6.2.2 PAN manager . . . 83

6.3 System power optimization . . . 85

6.4 Violin-pad . . . 87

6.5 Smart speech portal . . . 89

6.6 Information-capturing devices . . . 93

6.7 Related work . . . 94

6.8 Chapter summary . . . 95

7 Interface Cache 96

7.1 Interface cache . . . 97

7.2 Bluetooth-based cache-watch . . . 97

7.3 Evaluation . . . 102

7.4 Design issues . . . 104
7.5 Related devices . . . 107
7.6 Chapter summary . . . 108

8 Power Management Based on User Delay Prediction 109
8.1 The slow-user problem revisited . . . 110
8.2 Interaction modeling . . . 113
8.3 Psychology-based user delay models . . . 116
8.4 History-based user delay model . . . 119
8.5 Implementation of user delay prediction . . . 122
8.6 Theoretical analysis of DPM/DVS . . . 123
8.6.1 DPM and DVS techniques . . . 124
8.6.2 Theoretical energy saving and latency overhead . . . 124
8.7 Power models and DPM/DVS policies . . . 126
8.7.1 System power model . . . 127
8.7.2 DPM/DVS policies . . . 129
8.7.3 System implementation issues . . . 130
8.8 Experiments . . . 131
8.8.1 Experimental setup . . . 131
8.8.2 Experimental results . . . 134
8.9 Discussions . . . 143
8.10 Chapter summary . . . 145

9 Conclusions 147
9.1 Dissertation summary . . . 147
9.2 Limitations . . . 148
9.3 Future work . . . 149

References 150

List of Figures

2.1 System view of a mobile system 11

4.1 A hardware perspective of GUI energy consumption. 43
4.2 A software perspective of GUI energy consumption. 43
4.3 Looking for a file and creating an email. 44
4.4 Outline of the target for benchmark “Event loop”. The context uses the same code but with line 13 commented out. 49
4.5 Additional energy for showing texts of different sizes. 53
4.6 Additional energy for showing menu windows containing different numbers of items. 54
4.7 Additional energy for showing windows of different sizes. 54

5.1 Baseline power and additional hardware power consumption 65
5.2 Additional energy per word/letter for *Transcriber* 69
5.3 Ratio of energy consumptions for text output over speech output under different scenarios 71
5.4 Ratio of energy consumption for different text-entry methods over speech-based text entry 73
5.5 The maximal number of words per command for better energy efficiency 74

6.1 System overview of the PAN 80
6.2 Paging/Page-Scan session 81
6.3 Bluetooth power consumption 83
6.4 Software installed on iPAQ; the PAN manager is shaded 84

6.5 The violin-pad 87
6.6 Double button design 88
6.7 The smart speech portal 90
6.8 State-machine description of PIC I software 91

7.1 The prototype of a wrist-watch as the interface cache for iPAQ 98
7.2 Command format for the communication protocol 100
7.3 State-machine description of the cache-watch interface 102
7.4 Power consumption for the cache-watch in different modes 103
7.5 Cache-watch battery lifetime for different average communication intervals 103
7.6 Minimal frequency reduction for improving energy efficiency 105

8.1 Power consumption of Sharp Zaurus while running $Calculator$. 110
8.2 Interactive application: eventloop and event handlers 112
8.3 An STD for the Qtopia $Calculator$. 114
8.4 A menu window from the Qtopia application $Contact$. 118
8.5 User delay statistics for State $Number$ of $Calculator$ for User 1. 120
8.6 Triangle distribution of user delays for a state 127
8.7 Power modes and mode transitions. 128
8.8 Energy savings based on predicted user delays 136
8.9 Percentage of lazy errors and energy savings 137
8.10 Tradeoff between percentage of lazy errors and energy savings for User 1. 139
8.11 Tradeoff between percentage of lazy errors and energy savings for User 2. 140
8.12 Energy saving changes with mode power and transition time 141
8.13 Energy saving changes with mode power and transition time 142

List of Tables

3.1 Typical text-entry speeds for different methods 38

4.1 GUI nomenclatures 41
4.2 Hardware and software information on mobile systems 45
4.3 Benchmarks 48
4.4 Coding and compilation information for benchmarks 49
4.5 System energy breakdown for mobile systems 50
4.6 Energy characterization in EUs for different GUI platforms 52
4.7 Energy breakdown for presenting screens of different colors 55
4.8 Energy for different colors for the QPE theme on Zaurus 56
4.9 Additional energy for showing and hiding windows of different colors on a black background 57
4.10 Different color patterns on iPAQ1 58
4.11 Additional energy for different input methods 59

5.1 System information for iPAQ and Zaurus 63
5.2 Power consumption for different auditory outputs 67
5.3 Additional energy consumption for inputting a letter 69

7.1 Memory cache vs. interface cache 97

8.1 Percentage of system time and energy spent waiting for user input 111
8.2 Number of choices for different user interface features used in the Hick-Hyman Law 118

8.3 Parameter values for the psychology-based model 132
8.4 Actual user delay statistics 133
8.5 Tasks for usage trace collection 134

Chapter 1

Introduction

This dissertation addresses energy efficiency of mobile systems. Energy efficiency is not a new concern. Both academia and industry have made significant progress toward energy-efficient computing from both the hardware and software perspectives. However, for mobile systems, on which most applications are interactive, a new perspective is needed. This dissertation is devoted to such a perspective, namely, the *user's perspective*.

In this chapter, we first survey mobile systems in Section 1.1. Then we highlight the energy efficiency challenge for them in Section 1.2 and discuss how one can meet this challenge from the user's perspective in Section 1.3. We describe the dissertation contributions and provide its overview in Section 1.4.

1.1 Mobile systems: The future of personal computing

Mobile systems are personal computing systems much smaller than laptops. They provide a different set of services from what personal computers (PCs, desktops and laptops) do. We first outline their recent history, and then survey the computing and interfacing issues.

1.1.1 The convergence of evolution

Mobile systems have been evolving along three different tracks. The first track is personal *computing*, primarily including personal digital assistants (PDAs). Such miniature computers provide a subset of PC services as a natural extension to the form factor of laptops. Due to the much lower user productivity[1] possible on them, only personal information management (PIM) applications have seen some success. In fact, sales of PDAs[2] have been declining globally in recent years (2001-2004), according to IDC, a market research firm [81]. The second track is personal *connectivity*, including internet connectivity and voice communication. Cell phones are a prominent example that have penetrated all walks of life. By the middle of 2004, there were already 1.5 billion cell phone users around the globe, according to the International Telecommunication Union [104]. The sales of cell phones are enjoying a 29% annual increase at the time of this writing. The third track is personal *entertainment*, including games and media. Mobile game consoles, such as Nintendo's *Game Boy* and Sony's PSP, and mobile media players, such as Apple's iPod and Sony's Walkman, have seen considerable market success.

Mobile systems used to evolve along these tracks separately. Recently, however, there has been a significant convergence, especially in cell phones [41, 147]. More and more cell phones now provide PDA-like services such as PIM and productivity. Such systems are often called PDA phones or smart-phones. For example, the popular BlackBerry smart-phones provide voice communication and email services on the same system. Cell phones with digital cameras, or camera phones, have already become popular. Motorola is releasing cell phones that play music like the Apple iPod. Although a full convergence is yet to be welcomed by users [73] due to the increased cost per system, it is inevitable in the long run because hardware costs drop constantly. Mobile systems with such a convergence will be very promising candidates for serving the needs of future personal computing[3] . Mobile systems, especially smart-phones, seem to be now on an evolutionary track to become the single powerful system that people will carry with them everywhere.

[1] User performance or what a user can do in a unit time.

[2] The BlackBerry smart-phones from Research in Motion, Ltd. are excluded.

[3] Here and thereafter, we use "computing" to refer to computing, storage and connectivity.

1.1.2 The blessing of computing and curse of interfaces

Mobile systems always lag behind PCs in computing capacity. However, they benefit from the same technological progress that enables PCs to become increasingly powerful. In some sense, mobile system designers are able to learn many lessons from PCs to enable more efficient computing. While PCs are integrated at the board level, most mobile systems employ systems-on-a-chip (SoCs). While most PCs use processors based on the less efficient CISC architecture, most mobile systems use processors based on the more efficient RISC architecture [85]. While PCs have just begun to use multi-core architectures, handsets have enjoyed a dual-core architecture for a long time. Moreover, SoCs used in mobile systems usually have better support for power-saving mechanisms, such as dynamic voltage/frequency scaling and power management. With the massive storage capacity available from the micro drive technology [87] and high-speed connectivity from the third generation cellular networks, WiFi and Bluetooth, mobile systems are enjoying the same, if not more, computing capacity that our PCs enjoyed 5-10 years ago.

Unfortunately, user productivity, e.g., text entry speed, on mobile systems is still much lower than that on our PCs 10, or even 20, years ago. The main reason is that their size stays unchanged. Their small size has prevented them from enjoying a full-size display and keyboard, which are key to user productivity on PCs. This curse of interfaces not only prevents them from providing more services to users, but also imposes a great limitation on their energy efficiency. Therefore, we briefly survey interfacing technologies for mobile systems next.

Graphical user interfaces: Graphical user interfaces (GUIs) have become the basic software mechanism for human-computer interaction. Most GUIs are developed using a toolkit based on a certain platform available to the corresponding operating system (OS). For example, both PalmOS and Windows CE provide their own GUI platforms and toolkits. Linux-based mobile systems can either use the Qt/Embedded or X Window system, enjoying more choices of toolkits. Except those for PalmOS, all platforms and toolkits were originally designed for PCs and later ported to mobile systems. Therefore, many desktop vestiges can be found in the GUIs for mobile systems [213]. They may negatively affect energy efficiency and user productivity.

Stylus and touch-screen: The stylus and touch-screen duo has been widely accepted as the

default hardware interfacing mechanism on PDAs [68,190]. Along with the touch-screen, the stylus can be used as a finger to interact with GUIs. However, by converting the hand into one finger, such an interfacing method wastes our most powerful interaction tools, i.e., our fingers. It is no wonder that virtual (software) keyboard-based text entry is much slower. The stylus can also be used like a pen. There have been many text entry methods based on drawing or handwriting with a stylus. Unfortunately, we draw or write much slower than we type. Suffering from these severe inherent limitations, the stylus and touch-screen duo has received little welcome from smart-phone users, as a recent survey indicated [73].

Display: Although mobile systems can only afford very small displays, they do enjoy the same technological progress in displays that laptop PCs do [115]. Displays for mobile systems and laptops have been based on different liquid-crystal display (LCD) technologies [113]. Since LCDs are not light-emitting, external illumination is required, in the form of either ambient light or front/back lighting. The LCD and its lighting are notorious for being among the largest power-consumers in a mobile system. On the one hand, having a much smaller display than a laptop hurts user productivity. On the other hand, since it is easier to produce smaller displays with new display technologies, mobile systems have been at the forefront in employing them. The most recent example is the organic light-emitting diode (OLED) technology [58]. Since OLED-based displays obviate the need for external lighting, they can be much more power-efficient than LCDs. They have already appeared on cell phones and digital cameras, and are about to debut on PDAs. With plastic substrates, the OLED technology also promises a flexible or foldable display [59] that can be handled like a piece of paper.

Keyboard and keypads: Foldable full-size keyboards for mobile systems have been available for a long time. For example, HP offers a Bluetooth foldable keyboard for its popular iPAQ PDAs. Foldable full-size displays and keyboards may eventually enable mobile systems to take over most services from laptops. However, mobile systems are not just replacements for laptops. They provide many new services because they are immediately accessible and ready to serve with little space requirements. Unfortunately, setting up a foldable display/keyboard takes time and space, which eliminates such an advantage. Therefore, the curse of interfaces will not be dispersed by such

convertible interfaces. On the other hand, popular manual text entry methods suffer from a very low speed, such as multi-tap (used on cell phones), thumbing (used on mini-qwerty keyboards on PDAs), Thumbscript [206] and Fastap [46]. They share the same two problems. First, they are physically attached to the computer. The user has to hold the computer to type. Second, the user can at most employ two fingers for typing. Chording-based keypads [137, 209] do make full use of fingers. Nevertheless, they enjoy little market success because chording imposes a significant memory load on the user.

Handwriting/speech recognition-based input: Handwriting recognition based text entry has been available for a long time in various forms on mobile systems used for personal computing. The speed is now limited by the human writing speed and handwriting recognition accuracy, instead of by the time taken to perform handwriting recognition. Unlike the keyboard input, software or hardware, handwriting with a stylus makes a continuous demand on the processor, leading to poor energy efficiency, as we will see in this dissertation. Handwriting recognition based input requires the most resources, in terms of the number of fingers and energy, yet delivers the lowest productivity [234]. As mini hardware keyboards are making their way to high-end mobile systems, handwriting recognition is becoming a secondary choice. On the contrary, speech recognition based input first appeared on cell phones for command and control, e.g., voice dialing, and is gaining ground for use on PDAs. Due to the relatively limited computing capacity of mobile systems, speech recognition on them is likely to remain command and control-oriented, and domain-specific in the near future, like Voice Command [145] from Microsoft. Although speech recognition-based command and control requires higher power consumption, it does not require the display or stylus. As its accuracy improves, speech recognition based input will become very energy-efficient [234].

1.2 Energy efficiency and the slow-user problem

The curse of user interfaces is not the only challenge that mobile systems have to face. Moore's Law predicts that computing capacity doubles every 18 months in terms of the number of transistors per integrated circuit (IC). As the computing capacity of mobile systems increases, users are spending more time with them and relying on them for increasingly diverse services. Unfortunately,

battery capacity in terms of volumetric or gravimetric energy density is increasing at a much slower pace, about 5-10% annually for Li-ion batteries. Thus, energy efficiency is another challenge that mobile systems have to face. More importantly, tackling the curse of user interfaces is essential to meet the challenge of energy efficiency, as we will see soon.

Improving energy efficiency is the main focus of this dissertation. The specific approach this dissertation takes stems from the observation that there is no Moore's Law for human capacity growth. While processors become faster and interfaces become fancier, we, as human beings, still have to read the display, make a decision, and physically move during interaction at the same speed that the previous generation did. Such a speed gap leads to a phenomenon that we call the *slow-user problem*. That is, *an increasingly fast and power-hungry computer spends most of its time and energy waiting for a constantly slow human user for interactive tasks*. Obviously, the curse of interfaces on mobile systems just exacerbates the slow-user problem.

If one looks at the power consumption trace of a mobile system engaged in human-computer interaction, there are flat valleys separated by major power peaks, corresponding to the system's responses to user input. In the valleys, the system waits for user input. The width of a valley is usually on the order of seconds, while the width of a peak is much smaller, on the order of milliseconds. According to our analysis [233], *a mobile system spends over 90% of its time and energy in the valleys waiting for user input*. Similar findings have been made in [53] for desktops. Moreover, mobile systems interact with users through their interfacing components, such as displays and speakers. Energy characterization work [234] clearly shows that *interfacing components consume a significant portion of total power*. Since how long the interfacing components stay active is determined by users, who are slow as compared to computers, power-hungry interfacing components make the slow-user problem worse.

In summary, mobile systems usually spend most of their time *idle* instead of computing, and *interfacing* is equally, if not more, important as computing with regard to power consumption. Therefore, the slow-user problem cannot be solved by efficient computing techniques, whether applied to hardware or software. Instead, it indicates that energy consumption of mobile systems is significantly affected by the user, who is usually ignored. To successfully tackle the slow-user

problem, we need to take the user into consideration and optimize mobile systems from the user's perspective.

1.3 Energy efficiency: The user's perspective

Why is energy or power consumption important? From the user's perspective, the concern is not really the power consumption, energy consumption, or even battery lifetime. Instead, it is what the user can do, given the battery lifetime. Energy efficiency is, therefore, better evaluated in terms of energy consumption per user task. At a higher level, one needs to evaluate

$$\frac{\text{User productivity}}{\text{Average power consumption}}$$

or

$$\text{(User productivity)} \times \text{(Power efficiency)}.$$

From such a perspective, user interfaces and human factors have a large impact on energy efficiency, simply because they determine not only the power consumption but also the user productivity. Most low-power research has focused on reducing the power consumption, given a computation or interactive task. It is equally, if not more, important to optimize the interactive task itself, i.e., reduce the interaction power, and improve user productivity. This, indeed, is the approach taken by this dissertation that includes user interfaces and human factors in low-power research.

From the user's perspective, energy efficiency is determined by the tradeoff between user productivity and power consumption. In this dissertation, we will show how user productivity can be improved without increasing power consumption much and how power consumption can be reduced without sacrificing user productivity much.

There is a larger context for optimizing mobile systems from the user's perspective. First, more efforts should be devoted to making computers more *useful* instead of powerful. It was found that computers are successful in doing things that are impossible for human beings, i.e., compute-intensive tasks, but not equally successful for helping to increase user productivity for simple interactive tasks [123]. As argued in [192], "the old computing was about what computers could do; the

new computing is about what users can do." Second, less efficient computing can be tolerated for higher productivity of designers, e.g., programming with high-level languages.

1.4 Dissertation contributions

Viewing energy efficiency from the user's perspective, this dissertation presents a comprehensive treatment of the slow-user problem using an interdisciplinary approach. It makes the following contributions.

- **Limits of interface energy efficiency**: The dissertation offers a theoretical analysis for interface energy requirements based on human sensory and speed limits. It highlights the importance of user interfaces and human factors in determining energy efficiency as compared to computing. It also provides a theoretical foundation for energy-efficient user interface design.

- **Interface energy characterization**: The dissertation offers a comprehensive energy characterization of GUIs and other interfacing methods available on commercial mobile systems. Based on the characterization, it offers a comparative study for their energy efficiency. It concludes that speech-based input has a great potential to become the most energy-efficient input method since we can speak at a much higher rate than we can write or type. This comparative study offers guidelines for energy-efficient user interface design.

- **Interface cache**: Motivated by the memory-cache theory, the dissertation shows how a low-power interface cache device can be used to handle simple interactive tasks outsourced from a host computer, and thus improve the overall energy efficiency. It saves energy essentially by bringing the interface energy requirements closer to the theoretical minimal without sacrificing user productivity much. The dissertation describes the design and prototype of a wireless wrist-watch as an interface cache device for serving a mobile system.

- **Novel interfacing paradigm and devices**: The dissertation discusses the system and hardware design for a Bluetooth-based personal-area network (PAN) of low-power wireless interfacing devices for improving the user productivity on mobile systems. The network consists

of a wrist-watch, single-hand single-tap multi-finger keypad, smart speech portal, and GPS receiver. In addition to providing interfaces individually, these devices can serve a mobile system in a synergistic fashion. Collectively, they provide the user with immediate and natural access to computing power, and enable more and better services.

- **Power management based on user delay prediction**: Since mobile systems spend most of their time in waiting for users, reducing power consumption during user delays will be most effective for improving energy efficiency. The dissertation proposes to predict user delays based on user interface information, human-computer interaction history, and theories from the field of psychology. It shows that such a delay prediction can be combined with dynamic power management (DPM) for aggressive power reduction.

The dissertation is organized as follows. It discusses related work in Chapter 2. It starts with the theoretical analysis of how human factors impose limits on interfacing energy/power requirements in Chapter 3. It then presents an energy characterization of GUIs on commercial mobile systems in Chapter 4. It provides energy characterization for other interfacing methods on commercial mobile systems and a comparative study for them in Chapter 5. It discusses the design and prototype of a PAN of wireless low-power interfacing devices in Chapter 6. It proposes the concept of a wireless interface cache device and presents a prototype design in the form of a wrist-watch in Chapter 7. In Chapter 8, it presents techniques to predict user delays during human-computer interaction for aggressive power management. It concludes and discusses future work in Chapter 9.

Chapter 2

Related Work

In this chapter, we discuss related energy analysis and optimization techniques. We first present a system view of a mobile system in Section 2.1 as the context. Then we survey related energy optimization techniques in Section 2.2, and their system support in Section 2.3. We discuss related energy analysis and characterization work in Section 2.4. In Section 2.5, we discuss works that consider user interfaces and human factors for energy savings. We summarize the chapter in Section 2.6.

2.1 System view of a mobile system

The energy efficiency of a mobile system is determined by three factors: the user, software, and hardware, as illustrated in Figure 2.1. The user is the ultimate consumer of hardware resources including energy, and has a significant impact on energy efficiency. Software includes an OS, various applications, and their user interfaces. Applications serve the user through user interfaces and incur hardware activities through the OS.

Major hardware components include the SoC (processor), memory (usually RAM), mass storage (either hard drive or flash memory), network interface, display, and other interfacing hardware. Energy optimization inside a hardware component through better material, mechanical, device, circuit, and architecture design [27, 222] is important. However, we will not discuss such work since this dissertation is not concerned with hardware design per se. Hardware components can also pro-

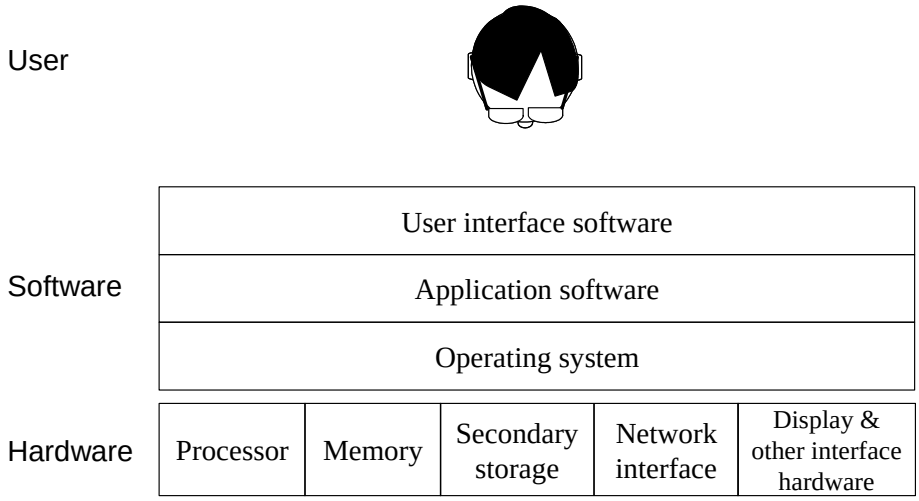


Figure 2.1: System view of a mobile system

vide power-saving mechanisms for software use through standard interfaces, such as the Advanced Configuration and Power Interface (ACPI) [2]. There are mainly two such mechanisms, power management and performance scaling.

Given the hardware, energy optimization usually reduces to hardware behavior optimization because energy is eventually consumed by hardware. There are three ways to optimize hardware behavior for energy savings. One can reduce the number of hardware activities because fewer hardware activities consume less energy. One can balance activities between different hardware components so that the overall energy consumption is reduced. Most importantly, one can change the hardware behavior to utilize the hardware power-saving mechanisms. Techniques optimizing hardware behavior in all three ways will be extensively surveyed in this chapter.

Since software, especially the OS, resides just above the hardware, its interaction with the hardware has been the focus of past research for energy efficiency. The user is involved only in generating software traces or through the use of abstract stochastic models. This dissertation, instead, focuses on the interaction between the software and user, especially user interfaces and human factors, for energy-saving opportunities.

2.2 Energy optimization

In this section, we survey techniques that optimize hardware behavior for energy savings. We address techniques that reduce hardware activity in Section 2.2.1. We discuss techniques that balance activities among different hardware components in Section 2.2.2. Since techniques that utilize the hardware power-saving mechanisms have been the most effective and popular, we cover them in Sections 2.2.3 and 2.2.4 under power management and performance scaling, respectively.

2.2.1 Hardware activity reduction

If software can finish the same computation with fewer instructions, it consumes less energy. From this perspective, many software performance optimizations done by compilers [3, 4, 6, 36, 151] save energy. If user productivity can be increased without much power overhead, energy efficiency will be improved as the product of user productivity and power efficiency. From this perspective, many user interface techniques that enhance user performance also improve energy efficiency [7,191]. Vallerio *et al.* investigate how GUIs can be designed to improve energy efficiency of mobile systems in [213]. When evaluating the impact of such performance improvements on energy efficiency, we must recognize that they save energy not only in "computing" components but also in power-hungry interfacing components, such as displays. Since they are out of the scope of this dissertation, we will not discuss these techniques further except briefly in Section 2.2.2. Instead, we focus on a group of techniques that reduce software functionality to reduce hardware activity for saving energy. They are called *software adaptation* techniques.

For data-intensive applications, hardware activity can be reduced by reducing data fidelity. Lower data fidelity means less data processing and hardware activity. The Odyssey system [56, 57, 158] provides OS support for applications to reduce input data fidelity. Data fidelity is application-specific, e.g., vocabulary size of speech recognition and video quality for a video player. The Puppeteer system [44] scales data fidelity to a higher level based on input components. In [13], Bharghavan and Gupta discuss an application adaptation framework. In [187], Shenoy and Radkov propose to transform the requested network data stream to reduce receiving and decoding energy. In [155], Narayanan and Satyanarayanan provide examples of computation fidelity reduction for

mobile applications. Mohapatra *et al.* investigate how video quality can be sacrificed for power savings in [148]. All these techniques are targeted at a single application. In [42], Efstratiou *et al.* propose a platform to coordinate adaptation of different applications to avoid competition for hardware resources. In [47], Fei *et al.* propose and implement a user-level coordination framework to adapt multiple applications for energy savings. The framework employs a priority-based pre-emption policy to coordinate running applications. By monitoring the energy supply and demand, it is able to select the right trade-off between energy conservation and application quality of service (QoS).

2.2.2 Hardware activity balance

We have discussed techniques that reduce software functionality to reduce hardware activity in the previous subsection. In this subsection, we discuss techniques that balance activities among different hardware components to reduce the overall energy consumption.

Computation and interface: Many techniques that improve software and user performance, as discussed in Section 2.2.1, save energy by balancing the energy consumption of the computation and interface. Suppose a performance optimization technique can reduce the task time by a fraction α, while increasing the power consumption of some hardware components by a fraction β. These hardware components are usually "computing" components, such as the processor and memory. Other hardware components, such as the display, are unaffected. Let γ denote the fraction of total power consumed by those affected hardware components before optimization. The optimization will save energy if and only if

$$(1-\alpha)\cdot\{(1-\gamma)+(1+\beta)\cdot\gamma\}<1$$

or

$$\alpha+\frac{\alpha}{\beta\cdot\gamma}>1.$$

A simple corollary is that the optimization will save energy if

$$\alpha > \beta \cdot \gamma.$$

For handhelds, γ is typically smaller than 0.5. Thus, the task time decrease only needs to be half of the power increase in the "computing" components for the technique to save energy.

Computation and communication: There are two seemingly opposite, but actually complementary, ways for tradeoffs between computation and communication energy consumption. On the one hand, computation offloading techniques [126, 162, 181] outsource compute-intensive tasks from a mobile system through a network interface to a wall-powered computer. With extra energy consumed in the network interface, these techniques save more energy in "computing" hardware components. On the other hand, Barr and Asanovic [8] find that overall energy consumption can be reduced by compressing data before transmitting them through the network interface. Anand *et al.* [5] propose to determine whether to retrieve files from the local hard drive or through the network interface based on the power-saving modes they are in.

Techniques surveyed so far are mostly effective for compute-intensive tasks. As we mentioned in Chapter 1, however, a mobile system is likely to spend most of its time and energy in idle periods instead of computing. Even worse, systems are usually designed for peak performance, although the majority of applications do not require peak performance. This leads to energy waste in unnecessarily powerful hardware components most of the time. Many hardware components, therefore, provide power-saving mechanisms for software, especially the OS, to tackle these two problems. For idle periods, there is the *power management mechanism*; and for unnecessarily high performance, there is the *performance scaling* mechanism. We next survey relevant techniques.

2.2.3 Power management

Power management [11] reduces the hardware functionality to save power. It stops the clock or powers off subcomponents after saving the operational context. As a result, a power-managed hardware component may not be fully functional and is said to be in a *power-saving mode*. It has to restart the clock or power-on corresponding subcomponents, and restore the saved operational context to exit the power-saving mode before restoring functionality. Such a process takes time,

which is called the *exit latency*. A hardware component can provide different power-saving modes with different combinations of power saving and exit latency. Typically, more the power savings, longer the exit latency.

Industrial support for power management has been surveyed in [116]. ACPI has become the de facto interface for a hardware component to export its power-saving modes. A system needs to determine when to put a hardware component into which power-saving mode, and for how long to avoid performance degradation due to exit latencies. The key problem is hardware usage prediction, especially to know when and for how long a hardware component will be idle. Four different approaches have been investigated for this purpose.

Greedy approach: The *greedy* approach makes no prediction at all. A hardware component enters a power-saving mode immediately after becoming idle; it wakes up when it receives a request. This approach limits itself to power-saving modes with relatively short exit latencies to avoid performance degradation. Its power-saving capability is thus limited. Due to its simplicity, however, this approach has been widely adopted by the industry and has been implemented in many OSs. Intel's QuickStart technology [101] puts a processor into a power-saving mode even between two keystrokes. The Linux kernel puts an idle processor immediately into a power-saving mode when there is no process running. Similarly, the processor of the IBM Linux watch system [110] enters its STANDBY mode whenever possible and wakes up upon user input. The STANDBY mode has an exit latency longer than 200ms, which is obviously perceptible to the user and is thus unacceptable for most applications on mobile systems. Nevertheless, IBM researchers consider it acceptable for applications on the Linux watch. In [16], Brakmo *et al.* implement a technique, called μSleep, that uses the SLEEP mode of Intel SA-1100 with an exit latency of about 10ms to reduce idle time power consumption. Fortunately, the latency is well below the human-perceptual threshold.

Time-out approach: The second approach is *time-out*. That is, a hardware component enters a power-saving mode when it has been idle longer than a threshold period of time. Usually, longer the idle time, deeper the power-saving mode. The hardware component wakes up upon receiving a request as in the greedy approach. Also like the greedy approach, time-out is simple and widely used. In [125], Li *et al.* investigate time-out power management for a hard drive using field-

collected traces. They find that more energy can be saved with a much shorter threshold (2 seconds) to spin down a hard drive than used by the industry then. They point out that hard-drive spin-down introduces user-perceptible latencies. They also find that the addition of a hard-drive cache does help save energy. Douglis *et al.* [39] propose to adapt the time-out threshold according to user's access patterns and priorities. Helmbold *et al.* [84] employ machine learning techniques for adapting the time-out threshold based on recent hardware activity. In [118], time-out power management for a hard drive with only one power-saving mode (spin-down) is formulated as a *rent-to-buy* problem. Putting a hard drive into the spin-down mode corresponds to buying. Irani *et al.* [105] extend the analysis of [118] to multiple power-saving modes. The deterministic online algorithm discussed in [105] determines for a hardware component which power-saving mode to enter based on how long it has been idle. They show that such a time-out strategy is two-competitive, i.e., the energy consumption is at most two times the minimal. They also propose to learn the distribution of idle periods online and adapt the time-out thresholds for entering different power-saving modes. The effects of exit latencies thus introduced are later studied in [180]. The time-out approach is typically used with power-saving modes with relative large power savings and long exit latencies, complementary to the greedy approach. Although widely used, it is unable to exploit short idle periods such as those that occur during human-computer interaction.

Predictive approach: The third approach is *predictive*. That is, the length of the next idle period of a hardware component is predicted based on its history. It is then put into the proper power-saving mode immediately. The hardware component can be woken up before the predicted idle period elapses or by a request. In [69, 70], Golding *et al.* extensively survey and study general techniques for predicting idleness in a computer and its components. All the techniques addressed are based on a model in which a hardware component receives requests with different statistics. Golding *et al.* also discuss the application of idle period prediction to hard-drive power management. The same philosophy is employed in [196]. Srivastava *et al.* use two heuristics based on the history of a hardware component's busy and idle periods to determine whether the next idle period will be long enough to justify a shutdown. The first heuristic is based on a linear or quadratic regression model. The second heuristic is simply based on whether the previous busy period is longer

than a threshold. Srivastava *et al.* obtain these heuristics through an analysis of the durations of busy and idle periods in collected traces. They acknowledge that there is a noticeable degradation in interactive performance if the exit latency is long but do not investigate how such a degradation can be avoided. In [95], the same approach is used but with an exponential-average prediction equation. To avoid performance degradation, Hwang and Wu attempt to wake up the power-managed hardware component before the next busy period. In [233], we use user interface information to predict idle periods during human-computer interaction for power management. By waking the system up before the next user input, we can make different tradeoffs between power savings and user productivity degradation.

Stochastic modeling approach: The most sophisticated approach is based on *stochastic modeling*. Most techniques with this approach are based on the following model. The hardware component to be power-managed is regarded as a *service provider*, which has a collection of states corresponding to power-saving modes. The requests for it are assumed to be generated by a *service requester*. The requests can be buffered in a *service queue* when the service provider is in non-operational modes. In [10], Benini *et al.* use a discrete-time controllable Markov process for modeling the service provider, a discrete-time Markov process for the service requester, and a discrete-time stationary controllable Markov process for the service queue. Based on these models, the power management problem is reduced to estimation of different parameters of the models and a stochastic decision process that minimizes a given cost formulation. Qiu *et al.* [172] extend this approach by using the continuous Markov process for the service provider, using two service queues of different priorities, and using the Poisson process for the service requester. The Poisson process model of the service requester makes the system event-driven. In [32], Chung *et al.* extend the model of the service requester and employ online adaptation to handle unknown or non-stationary workload. Šimunić *et al.* [210] extend the event-driven model of [172] by using a time-indexed semi-Markov process. Moreover, they find that the Poisson process cannot appropriately model service requesters in reality. For a Poisson process, event arrival intervals have an exponential distribution. On the contrary, Šimunić *et al.* find Pareto distributions model these intervals better when the service queue is empty. They correct the misuse of exponential distributions for other model

components based on their observations of real systems as well. They also propose to formulate the power management problem using *renewal theory*, which is simpler and less versatile than Markov processes. A stochastic modeling technique usually gives the optimal solution when its assumptions about the statistics of the service provider, requester, and queue are true. Unfortunately, in reality, not all these assumptions are always true, leading to an inferior power management policy. Moreover, the stochastic modeling approach is much more complicated than the other three approaches. It has been greeted without much enthusiasm by the industry so far.

Opportunity enhancement: Besides the above four approaches for determining how to power-manage a system during idle periods, researchers have also investigated techniques to increase the size of idle periods. Lorch and Smith propose OS techniques in [129] to detect whether a processor is executing useful work and expose its idleness for power management. Shih *et al.* [188] investigate how a low-power and low data rate radio module can be used as a radio-activity monitor. The power-hungry main wireless module can be put into power-saving modes most of the time and can be woken up by the monitor. In [83], Heath *et al.* propose application transformations to increase the idle time of a peripheral device.

Buffering (caching) the requests or input data to a hardware component can extend its idle period. Bertozii *et al.* [12] propose to buffer streaming multimedia data so that the network interface can be power-managed without interrupting a continuous playback. In [164], Papathanasiou and Scott study OS hard-drive caching and pre-fetching for increasing hard-drive power management opportunities. Cai and Lu [24] investigate buffering as a general mechanism to increase power management opportunities. They formulate the buffer management problem as an *inventory control* problem. Buffering is also used for performance scaling, as will be discussed later.

When multiple service providers serve multiple requests, properly scheduling the serving of these requests may also extend the idleness of the service providers. This has been investigated by many researchers. We will discuss it further when we address system support for energy optimization. Since the idle periods that a scheduler can affect are usually very short, scheduling is a more powerful tool for performance scaling than for power management.

2.2.4 Performance scaling

Unlike power management, performance scaling does not affect the hardware functionality but changes the performance. Although it usually saves less power than power management does, it can be applied to more scenarios because there is no functionality change.

General performance scaling: "Performance" has different meanings for different hardware components. For ICs including processors, performance means speed or clock frequency. Supply voltage reduction is very effective in saving both energy and power consumption [21]. Because a lower supply voltage incurs a longer gate delay [88], *supply voltage scaling* is usually combined with *clock frequency scaling* or *speed scaling*. Frequency reduction itself only saves power but not energy for the affected hardware component. Moreover, leakage power will be significant in future technologies. Increasing the gate threshold voltage will reduce leakage power but increase the gate delay. Thus, *threshold voltage scaling* through adaptive body biasing (ABB) [119] can also be combined with supply voltage and frequency scaling [74, 111]. The tradeoff is not only between speed and energy consumption but also between switching and leakage energy consumption.

For hard drives, performance means data rate. The spindle motor of a hard drive can rotate at different speeds [80]. Lower the speed, lower are the date rate and power consumption. For displays, performance can be color, contrast, or brightness. The luminance of a pixel on an OLED-based display can be individually controlled. By darkening the pixels outside the window of user focus, one can reduce the display power consumption without sacrificing its usability much [107]. Techniques have been proposed to scale the absolute luminance for power savings without sacrificing the relative luminance based on screen contrast [28, 29, 61].

Processor performance scaling – theories and simulations: The majority of performance scaling research is focused on processors, which provide well-defined and flexible support for scaling. Performance scaling can utilize processor idle periods that are too short for a processor to enter a power-saving mode. By reducing *processor* performance to remove these idle periods, one can save energy for a given computation without sacrificing *system* performance, e.g., as perceived by the user. If execution delay is tolerable, more energy can be saved by reducing processor performance further.

All performance scaling techniques are concerned with obtaining information about processor idleness and managing delay introduced by performance reduction. Weiser *et al.* [217] are the first to study processor performance scaling for energy reduction. They rightly point out that clock frequency reduction must be combined with supply voltage reduction to save energy. Henceforth, in many works that follow, determining the appropriate processor performance level to finish a task is called *voltage scheduling*. Weiser *et al.* evaluate energy efficiency in terms of *million instructions per Joule*. They propose an interval-based algorithm, PAST, that uses the processor activity in the last interval to predict that of the current interval for performance scaling. However, their conclusion that "it is better to spread work out by reducing cycle time (and voltage) than to run the CPU at full speed for short bursts and then idle" may not always be correct. Based on implementation and measurements, Miyoshi *et al.* [146] find that for some processors it consumes less energy to run as fast as possible (i.e., no performance scaling) and then enter a power-saving mode. It is very important to balance the opportunities for power management and performance scaling.

Govil *et al.* [76] extend the work in [217] with more sophisticated prediction heuristics. They reluctantly conclude that "simple algorithms based on rational smoothing rather than 'smart' predicting may be most effective" [76]. Pering *et al.* [165] further extend these two works with traces from PDA applications, including an interactive one. Their contribution primarily lies in breaking the processor activities into application-specific events, instead of fixed-length intervals. Each event is associated with a deadline, which is also application-specific. For example, an event for an interactive application is defined as from the start of a user input to when the computer finishes processing it. The deadline is chosen to be $50ms$, which is below the human-perceptual threshold for causality [109]. Only when an event is finished after the deadline, a penalty is imposed. Such a method obviously benefits from extra information from applications so that it can slow down the processor in a better way. It is, however, hard to automate and critically dependent on the quality of event definitions. Pering *et al.* present the same technique with more processor information in [166]. Ishihara and Yasuura present in [106] an elegant theoretical analysis of power-delay optimization for determining the supply voltages for a deadline-bounded task. They assume the frequency is scaled according to the supply voltage. They also offer an *integer linear programming* (ILP) formulation

for determining the optimal supply voltages for multiple real-time tasks with hard deadlines. Qu extends such a theoretical analysis in [176] with different constraints on supply voltages.

In [130], Lorch and Smith make a very important observation on the tradeoff between computation and interface energy consumption, as we discussed in Section 2.2.2. They further conclude that "one should reduce the voltage only when it will not noticeably affect performance." This conclusion is a precursor of our definition of energy efficiency from the user's perspective. Lorch and Smith also find that a continuously increasing supply voltage instead of a constant one for a deadline-bounded task consumes minimal energy if the time distribution of its processor workload is pre-known. Based on this conclusion, they propose an algorithm for processor acceleration to conserve energy (PACE). The algorithm modifies any voltage scheduling algorithm as proposed in [76, 165, 217]. Since no processor can scale its supply voltage continuously, Lorch and Smith have to change the supply voltage only at discrete time points. Moreover, since the processor workload is usually unknown *a priori*, they have to rely on sampling and distribution estimation techniques. Using benchmarks including interactive applications, they show that PACE can improve most voltage scheduling algorithms, but with an unrealistic assumption that the supply voltage can be any value between the minimum and maximum. PACE also assumes there is only one process running at a time, which is, again, unrealistic for most application scenarios for modern mobile systems. Moreover, the results are from simulation of collected traces. For a real system, such frequent supply-voltage changes, as required by PACE, may incur significant performance and energy overhead in addition to bringing up many other system issues. When Lorch and Smith report their implementation of PACE in [128], they conclude that PACE would not save energy for processors then available.

Processor performance scaling – implementations: Burd *et al.* report their implementation of a dynamic voltage-scalable processor in [20, 21] and the corresponding system implementation of a voltage scheduler in [165, 166]. In [79], Grunwald *et al.* implement the voltage scheduling algorithms from [76, 165, 217] on a COMPAQ Itsy Pocket computer. The Itsy system has an Intel StrongARM SA-1100 SoC and runs the Linux/X Window system. The SoC runs with a clock frequency between 59MHz and 206MHz according to software specification. They manage

to run the SoC with two supply voltages, 1.23V and 1.5V. They find that processor behavior is very hard to predict, especially for interactive applications. They conclude that all these proposed algorithms [76, 165, 217] work in non-desirable ways and make the Itsy system less responsive. They also observe nonlinearity in the relation between processor frequency and idleness, mainly due to the processor-memory bottleneck. The impact of such system issues on energy saving through performance scaling is investigated in depth by Martin and Siewiorek in [141] based on measurements of an Itsy system with scalable frequency. Again, these works illustrate how theoretical analysis and simulation can differ from results based on real implementations. Interestingly, the processor-memory bottleneck is the basis for compiler-directed performance scaling in [92], as we will discuss later in Section 2.3.

Commercially, Intel's Pentium® III Mobile processor [103] debuted with the SpeedStep® technology [102], and Transmeta's Crusoe^{TM} processor with the LongRun technology [208]. A Pentium® III Mobile processor with the SpeedStep® technology offers only two performance levels: one for maximum performance and the other for optimized battery lifetime. The processor can change levels according to user choice or power-supply status, i.e., wall-power or battery. Transmeta's LongRun^{TM} technology offers more performance levels. The frequency and supply voltage of a Crusoe^{TM} processor can be adjusted stepwise (33MHz for frequency and 25mV for supply voltage) according to the performance requirement [54]. Notably, the LongRun^{TM} technology is implemented as firmware on the processor. There is no need for any OS support. As opposed to the taxonomy used in this dissertation, Transmeta regards performance scaling as a power management technology.

Flautner and Mudge realize the limitations of application-dependent techniques studied in [165]. They propose an episode-based algorithm in [52] and implement it in the Linux kernel for a Sony laptop with a Crusoe^{TM} processor [51]. An episode resembles an event in [165]. However, an episode is application-independent and is automatically detected as either interactive or periodic. Flautner and Mudge regard the duration of an interactive episode as the latency for the computer to respond to a user input. The start of an interactive episode is noted when the X server sends a socket message to another process. The X server, receiving process, and all other processes com-

municating with them form the *task set* of the episode. When the behavior of all processes in the task set meets certain criteria, the episode is regarded as finished. A periodic episode is detected by monitoring processor execution. To guarantee computer responsiveness, processor performance is scaled according to the interactive episode duration. Unfortunately, the lack of cooperation from the Windowing system imposes severe limitations. First, Flautner and Mudge have to modify many system calls and kernel modules to implement their episode-detection mechanism based on inter-process communication via UNIX sockets. Second, the duration of a detected interactive episode by their method is different from the response time to a user input. For example, when the user is engaged with the foreground GUI application, another background GUI application can communicate with the X server. In this case, the task set will include the background GUI application, and their method to determine the end of an episode will not give the computer response time to a user input for the foreground application. As a result, their performance scaling techniques cannot guarantee computer responsiveness very well.

Lorch and Smith describe a processor performance scaling system for Windows 2000, called RightSpeed, in [128, 131]. The RightSpeed system implements PACE [130] and a task-specification interface for an application to convey information to the OS. The authors admit that few applications will abide by such an interface. For this reason, they implement a task detector to automatically obtain needed application information. A user interface event marks the start and type of a task, which completes when all non-idle processes are blocked and there is no I/O activity, or another user interface event is received. Such a detector is very similar to the interactive episode detection mechanism in [52], although it shuns the latter's complication. Like the interactive episode in [52], a task thus detected does not necessarily correspond to the processing of a user interface event. Although Lorch and Smith claim that the RightSpeed system reduces deadline misses for tasks, meeting a task deadline is different from keeping the computer responsive. Even worse, they find that PACE does not save energy for processors then available. Nevertheless, they find that different types of user interface events incur different processor loads. They successfully exploit this fact in the RightSpeed system [131].

In [220], Yan *et al.* propose to seek full cooperation from the windowing system to solve

the problem that techniques in [52, 128] face. Instead of using kernel activities to determine the computer response time, the authors use the X server to mark the start of a user event and use the event-loop mechanism of the Xlib to mark the end of a computer response. There is no kernel modification at all. With help from the windowing system, the computer response time is accurately and efficiently determined. Yan *et al.* also show the response time can be used to drive processor performance scaling to save energy while keeping the computer responsive. The techniques are implemented on a Pentium® IV Mobile based IBM ThinkPad laptop running Redhat Linux. More importantly, the techniques handle multiple running applications naturally.

Real-time and multimedia applications: Real-time and multimedia applications have been researchers' favorite for studying performance scaling. A real-time application usually provides task and deadline information that techniques discussed above sorely need. A multimedia application incurs periodic and even more predictable processor activity, and is in most cases a soft real-time application. Since this dissertation is not focused on either real-time or multimedia applications, we will survey these works only briefly.

Although a mobile system usually runs multiple processes even if only one application is engaged in user interaction, most performance scaling techniques discussed so far virtually ignore multiple processes and their scheduling due to diverse process characteristics. Only for real-time systems, process scheduling has become a practical and dominant mechanism for performance scaling [89, 90, 117, 124, 135, 136, 159, 168, 189]. Jha offers a very good survey in [108]. Kwon and Kim also provide a survey in a very recent journal article [120]. Among all these works, only Pillai and Shin [168] offer an implementation based on Linux.

Hughes *et al.* [93] study how architectural adaptation can be combined with performance scaling to save energy for multimedia applications. Lu *et al.* [132] and Im *et al.* [98] propose to use buffers at different stages of media processing for processor performance scaling. As discussed in Section 2.2.3, a buffering mechanism is also used to increase the opportunity for power management in [12, 24, 164]. Like the authors of [79], Pouwelse *et al.* [169] also manage to change the supply voltage of an Intel StrongARM SA-1100 process on their LART system. They point out that it is difficult to predict the media processing workload based on history and that inaccurate

prediction significantly reduces energy savings. While Choi *et al.* [30] use frame-based history to improve prediction for performance scaling, Pouwelse *et al.* argue that applications should explicitly convey their estimation of workload to the OS for performance scaling. They demonstrate how media processing programs can be modified to convey such information in [169–171]. A similar approach is reported by Chung *et al.* in [33]. Lu *et al.* [134] adopt a control system approach to performance scaling for frame-based multimedia workloads. In its essence, however, this approach just involves nonlinear history-based workload prediction. These approaches rely on application-specific workload estimation and require changes to the source code. Yuan *et al.* [223, 224] find that workload (demand) distributions for multimedia applications are usually very stable. They employ a simple histogram-based technique to estimate such distributions online, different from what is used in [130]. They then integrate voltage scheduling and process scheduling based on such an estimation in the GRACE framework.

Summary: Processor performance scaling techniques consists of three parts: *workload estimation, process scheduling, and performance setting*. Workload can be estimated in many different forms. *Processor utilization* in an interval and its different averages are used in [76, 217]. *Workload distribution* is used in [130, 223, 224]. *Task duration* is used in [52, 128, 220]. For real-time and multimedia applications, workload can be better estimated and deadlines are usually known. Although process scheduling is ignored by many researchers who assume only one process is running, it is critical for systems running multiple processes, especially deadline-bounded ones such as real-time and multimedia applications. Process scheduling is often integrated with performance setting [168, 223] in that a system determines which process to run with what performance. Since most commercial processors only offer a limited set of performance levels, performance setting usually has to use this limited set to approximate an algorithm that requires continuously scalable performance [130]. For a workload, estimated or pre-known, a fixed performance level can be selected [51, 76, 165, 217], or a selected series of performance levels can be applied consecutively [130, 132].

2.3 System support for energy optimization

We have so far surveyed related techniques for energy optimization. Most mobile systems, however, need to implement these techniques as part of the OS or application programs. System support from the compiler, middleware, and OS are therefore necessary. Indeed, the implementation-based works discussed above do implement some system support, but in a technique-specific fashion. We next survey general system support techniques from the compiler and OS worlds in Sections 2.3.1 and 2.3.2, respectively.

2.3.1 Compilation for performance scaling

Mosse *et al.* [149] were among the first to use compilation for performance scaling. They use a compiler to insert check-points into program source code so that workload information can be obtained at run-time for performance scaling. However, the compiler itself does not make any decision about performance scaling. Hsu *et al.* [91] were the first to use compilation to identify program regions for which performance can be scaled down. Hsu and Kremer summarize and advance their previous work on compiler-assisted performance scaling in [92]. Their most updated compiler technique uses the program profile to identify program regions in which the processor can be slowed down for reasons like the processor-memory bottleneck. The compiler inserts systems calls to change processor performance directly and off-line. The technique is implemented using the SUIF 2 open-source compiler system [200] and tested on a Linux-based laptop with the SPECfp95 benchmark. The SPECfp95 benchmark is compute-intensive but not very processor-intensive, as data exchange with the memory is significant. It provides significant opportunities for the proposed technique. As a result, one has to be careful when evaluating the technique for typical mobile applications. Xie *et al.* [219] explicitly consider the processor-memory bottleneck and performance scaling overhead while extending the theoretical analysis in [106, 182] to explore opportunities and limitations of compile-time performance scaling. Their technique requires a detailed execution profile of the program and solution of large mixed-integer linear programming problems. Moreover, the technique is validated by simulation instead of implementation. For example, the system overhead of performance scaling is more than the latency for a processor to change its performance

level because the performance scaling instruction from a running program has to go through the OS. All these compiler-assisted performance scaling techniques, unfortunately, suffer from a serious limitation: they are unable to handle multiple processes. As a program augmented by these techniques starts, it changes processor performance regardless of the processor load. Such a limitation is inherent in the technique because the program running context cannot be known at the time of compilation. Without interaction with the OS and its supports, compiler-assisted performance scaling techniques will fall short of their target.

2.3.2 OS support

The OS on a computer manages almost all the hardware resources. Most energy optimization implementations discussed in Section 2.2.3 and Section 2.2.4 are actually implemented as parts of the OS. In these implementations, the OS monitors the resource usage, makes predictions, and determines the power-saving mode or performance level for a hardware component. Yet these implementations do not manage the energy consumption of the whole system. We next discuss additional works that study general OS support for system-wide energy optimization.

Energy or battery lifetime seems like just another resource to be managed by the OS. In fact, it is very different from conventional hardware resources. First, energy is consumed through many hardware components. Second, while conventional resource management has focused on scheduling to avoid "contention," energy can be consumed by multiple hardware components simultaneously. "Contention" matters only when the battery discharge rate and thermal management are matters of concern. Third, while conventional resource management is concerned with better utilization of a hardware component for higher performance, energy management is more concerned with creating opportunities for hardware power-saving mechanisms, e.g., elongating idle periods. As a result, OS-based energy management is very challenging.

Neugebauer and McAuley extend their process-based resource accounting in the Nemesis OS to include energy consumption in [157]. They use the same pricing mechanism used for conventional resources to manage energy consumption. Such a direct extension actually ignores the three points made above, and is thus inadequate. In the Milly Watt project, researchers from Duke University

propose to regard energy as one of the most important resources that the OS manages [43,212]. They discuss their implementation, ECOSystem, in [228]. Like Nemesis [157], the energy management in ECOSystem is built upon process-based energy accounting with a *currentcy* model. A unit of currentcy is the right to consume a certain amount of energy within a certain period of time. Both the Nemesis OS and ECOSystem suffer from two fundamental limitations. First, they account for energy consumed by a process by how many hardware activities it incurs. However, the same activity often consumes different amounts of energy in a hardware component, depending on the context. For example, suppose process A writes 1KB data to the hard drive every one minute and process B writes 1MB data to it every one second. Process A will consume very different amounts of energy in the hard drive, depending on whether Process B is running or not. When Process B is running, the hard drive will stay in the run mode and Process A will consume insignificant extra energy. When process A is running alone, the hard drive will enter a power-saving mode between the accesses and the extra energy consumed by the hard drive due to process A will be significant, especially in view of the energy overhead for hard-drive mode switchings. Second, both Nemesis and ECOSystem fail to separate hardware information from the kernel because they implement power models for hardware components in the kernel for energy accounting. Consequently, the OS has to be customized for every different computer system it runs on. Due to these two limitations, their approach has not yet seen much popularity.

Lu *et al.* [133] take a much less aggressive approach. They use the exponential average of the access intervals of a process as the process-specific utilization of a hardware component. Then they aggregate utilizations from all processes, weighted by each process's share of the component, to obtain the component utilization. A hardware component is shut down when its utilization is lower than a threshold. They also schedule requests to a hardware component to increase the opportunities for power management. Their system, however, suffers from several limitations. First, they require the application to tell the OS when it needs a hardware component and the deadline by which such a need is satisfied. Although this automatically solves the most difficult problems, i.e., workload and deadline estimation, for power management and performance scaling, it is not easy to do in reality. Second, although they identify several parameters that affect the tradeoff between energy saving

and performance, there is no feedback mechanism for the system to balance them. In addition to these limitations, the system is not verified with real applications or traces. The workloads used in the study are unrealistic. Workload 1 only records user activities with long idle periods whereas workloads 2 and 3 are completely synthetic. Although Lu *et al.* emphasize that these techniques are targeted at interactive applications, user-perceived performance or computer responsiveness is not reported.

In [5], Anand *et al.* propose and implement interfaces between the power management module and device drivers so that a device can export its power model and current operational mode. They also implement application programming interfaces (APIs) for an application to retrieve device information related to power. Based on these interfaces, they implement middleware to balance the energy consumed by the hard drive and network interface. To minimize energy consumption by choosing from among alternative devices for the same service, they use a mechanism based on *ghost hints*. A ghost hint informs an application about how much energy would be consumed if a different device were used for the service. This work successfully minimizes changes to the kernel and separates concerns of performance and energy by implementing power models in device drivers. By using ghost hints to account for energy in the program execution context, it also addresses the issues raised above about managing energy as a resource. However, power management is not transparent to application developers, who have to customize, mostly manually, their source code according to the supported APIs.

Viewing the OS as only one layer of the system, another group of researchers proposes to provide cross-layer supports for energy optimization, i.e., to orchestrate the application software, middleware, OS, and hardware. Energy-aware software adaptation is a preliminary form of cross-layer solution, as discussed in Section 2.2.1. Flinn and Satyanarayanan [56, 57] extend the Odyssey platform to adapt input data fidelity for multiple applications based on predicted energy demand and targeted battery lifetime. The platform supports cross-layer energy optimization only when data fidelity reduction increases idle periods since only time-out-based power management is used. A significant cross-layer synergy is supported in the GRACE cross-layer adaptation framework proposed in [225]. The framework scales multimedia quality, OS services, and processor perfor-

mance simultaneously for energy savings according to user preferences. A similar work is reported in [148].

2.4 Energy analysis and characterization

By now, we have surveyed energy optimization and their system support techniques. In this section, we briefly survey works that offer insights into how energy and power are consumed in a system. Many works characterize power consumption by different hardware components of mobile systems. Power characterizations for laptops can be found in [127]. Energy characterization for the Itsy Pocket computer is presented in [45, 55]. That for a Palm Pilot is offered in [34]. Power characterization for Intel's experimental personal server, a handheld, is offered in [178].

Brooks *et al.* design and implement a processor power simulator, Wattch [18], based on the SimpleScalar simulator and data from Intel. Similar processor power simulators have also been implemented by others, such as SimplePower [221]. Wattch has been widely used in academia for studying architectural techniques for processor power reduction. It is widely known, however, that different power simulators often arrive at very different estimates [66]. Unfortunately, without accurate data and models from industry, researchers in academia have to rely on these simulators. For other hardware components, modeling is much easier since cycle accuracy is usually unnecessary. Stemm *et al.* [198] were the first to offer a detailed power characterization of several wireless network interfaces on handhelds. They point out that power management of the idle time is much more important than reducing data transfer for energy efficiency. Zedlewski *et al.* investigate hard-drive power modeling in [227]. They implement a hard-drive power simulator, called Dempsey, based on measurement [230].

There are also a number of software energy characterization works. Energy consumption of the OS, especially real-time embedded system OS, has been extensively studied [1, 9, 38, 201]. General software energy estimation and optimization techniques have also been investigated extensively [152, 194, 202, 207, 211]. Our work in [232] and [234] is the first to characterize energy consumption of user interfaces.

2.5 User interfaces and human factors

In this dissertation, we address energy efficiency from the user's perspective, and focus on the impact of user interfaces and human factors. Therefore, we briefly discuss related work that takes them into consideration in this section.

Interactive applications incur many computer idle periods. Our analysis [233] shows that a computer spends most of its time and energy in such idle periods for interactive applications. Many power management and performance scaling works [52, 79, 130, 165] use interactive applications as benchmarks. Nevertheless, these works treat them in the same fashion as compute-intensive ones without exploiting human factors to make resource usage prediction better. Some of them [52, 130] propose to use the human-perceptual threshold to slow down the system so that it can respond before the user can perceive the delay. They still focus on system busy time although most energy is consumed in the system idle time. In [131], Lorch and Smith find that different user interface events incur different computation loads for the processor. Therefore, they propose to conduct performance scaling based on user interface event information. Again, they target the system busy time. On the contrary, we investigate how user interface information can be used to predict the system idle time for power management in [233]. In [37], Dalton and Ellis propose to use sensors and cameras to detect user presence for power management. Although the energy overhead for their detection method is very high, such a method points to a new direction for making power management user-aware and context-aware.

Human users only have a limited visual field. Display areas outside the visual field present little information. In [107], HP researchers propose to darken display areas outside the window of user focus to save power. This is possible only for OLED-based displays, for which the luminance of individual pixels can be controlled separately. User studies of this technique are reported in [14, 82]. Unfortunately, luminance of individual pixels on LCDs cannot be controlled separately since current mobile computers use a single back or front lighting. Flinn and Satyanarayanan [56] propose to use multiple lightings, called *zoned backlighting*, to tackle this problem. Although the zoned backlighting is much simpler than the HP's pixel darkening technique, it has not yet seen much industrial acceptance.

2.6 Chapter summary

The energy efficiency of a mobile system is determined by its user, software, and hardware. Given the hardware, researchers have mostly looked at the interaction between software and hardware for energy-saving opportunities, especially those for hardware power-saving mechanisms. This chapter presented an extensive survey of techniques from such an endeavor. These techniques address energy efficiency from the perspectives of hardware and software. Many of them are effective only for compute-intensive or even processor-intensive applications. For more energy-saving opportunities on mobile systems, we need to look into the interaction between the user and system, and address energy efficiency from the user's perspective. As can be seen from this chapter, techniques considering user interfaces and human factors have been surfacing recently. The remaining chapters try to establish the efficacy of such techniques.

Chapter 3

Energy Efficiency Limits Imposed by Human Factors

Starting from this chapter, we are going to explore in depth the energy-saving opportunities for mobile systems from the user's perspective. Since our focus is on user interfaces and human factors, we examine how such factors impose limits on the energy efficiency of mobile systems, addressing limits on interface power/energy consumption and interaction speed in Sections 3.1 and 3.2, respectively.

3.1 Sensory perception-based limits

Landauer [122] showed that the theoretical minimal energy consumption of an irreversible logic operation is $kTln2$, where k is the Boltzmann constant and T is the temperature. kT is of the order of $10^{-21}J$ at room temperature. All commercially available computing devices use irreversible logic operations and are hence governed by this bound. On the other hand, the computer has to communicate with its human user through the latter's sensory channels. These channels in fact set the minimal power/energy requirements for the computer output.

3.1.1 Visual output

Human vision energy thresholds have been measured in different forms [22] in terms of minimal absolute energy, minimal radiant flux, and just-perceptible luminance. Minimal absolute energy is measured for a very small solid-angle field, e.g., a point source, presented for a very short time ($10^{-3}s$) so that no temporal summation of radiant flux occurs. Minimal radiant flux is measured for a very small solid-angle field lasting for a long time so that temporal summation of radian flux occurs. Just-perceptible luminance is measured for a large-area visual field. These thresholds are used to estimate the energy/power dissipation lower bound for displaying information as follows.

Minimal absolute energy: Let us assume the user's cornea area is A, viewing distance D, and viewing angle Ω. We assume the light irradiance is the same for every point within the viewing angle at the same distance from the point source. Let $E_{min}(\lambda)$ denote the minimal light energy reaching the cornea that is detectable by the user for light of wavelength λ. The total energy emitted by the source is thus:

$$E(\lambda) = \frac{\Omega D^2}{A_i} \cdot E_{min}(\lambda) \approx \frac{\Omega D^2}{A} \cdot E_{min}(\lambda)$$

where A_i is the area of the viewing sphere that is incident on the cornea. A_i is approximated as the cornea area A.

Experimental results reported by psychology researchers [22] indicate that E_{min} for light of wavelength $510nm$ is about $2 \cdot 10^{-17} \sim 6 \cdot 10^{-17} J$. Assuming $A = 0.5cm^2$, $D = 0.3m$, and $\Omega = 0.125 \cdot 2\pi\ sr$, we have $E \approx 3 \cdot 10^{-14} \sim 9 \cdot 10^{-14} J$, which is about seven orders of magnitude larger than the energy required for an irreversible logic operation.

Note that the energy limit derived above is for rod vision, which is the colorless human vision under extremely low luminance. Only the cone vision contains color and is normally required for human-computer interaction. The energy threshold for cone vision for $\lambda = 510nm$ is more than 100 times that of rod vision. For users to sense color, the minimal energy would thus be of the order of $10^{-11} J$.

Minimal radiant flux: Let $R_{min}(\lambda)$ denote the minimal radiant flux for light of wavelength λ that humans can sense. For the viewing distance D and viewing angle Ω, the source radiant power

is given by:

$$\Phi_{min}(\lambda) = \frac{R_{min}(\lambda) \cdot \Omega D^2}{683 \cdot V(\lambda)}$$

where $V(\lambda)$ is the relative visibility factor and 683 is the spectral efficiency for $\lambda = 550nm$ in $lumen/W$. According to [22], the minimal radiant flux for white light rod vision is about $4 \cdot 10^{-9} lumen/m^2$. Assuming $V(\lambda)$ for white light to be 0.8, we obtain $\Phi_{min} \approx 5 \cdot 10^{-13} W$ under the same assumptions for D and Ω as before.

Just-perceptible luminance: Suppose the just-perceptible luminance for light of wavelength λ is $L_{min}(\lambda)$. Let S denote the area of the display and Ω the viewing angle. The total display radiant power, $\Phi_{min}(\lambda)$, is then

$$\Phi_{min}(\lambda) = \frac{L_{min}(\lambda) \cdot S \cdot \Omega}{683 \cdot V(\lambda)}$$

For white light, L_{min} has been determined to be $7.5 \cdot 10^{-7} candella/m^2$ [22]. With the same assumptions as above, the minimal radiant power for a $12.1''$ laptop display and white light is about $5 \cdot 10^{-11} W$. For comfortable reading, the luminance level is, however, about $\frac{100}{\pi}\ candella/m^2$ [22], which requires a radiant power of about $2mW$ for a $12.1''$ display. This minimal radiant power for comfortable reading is about seven orders of magnitude larger than the just-perceptible threshold.

3.1.2 Auditory output

Let Ω denote the solid hearing angle and D the distance between ears and the sound source. The minimal sound intensity human beings can hear is about $10^{-12} W/m^2$ for a sound field of relatively long duration ($>300ms$) [63]. Below $300ms$, the threshold sound intensity increases fast as the sound duration decreases [63]. Therefore, we can estimate the minimal energy, E_{min}, for human beings to detect one bit of auditory information to be

$$E_{min} = 10^{-12} \cdot 300 \cdot 10^{-3} \Omega D^2$$

Assuming $\Omega = 0.125\pi\ sr$ and $D = 0.3m$, we have $E_{min} \approx 10^{-14} J$, which is of the same order of magnitude as the minimal energy required for displaying one bit of visual information. Note that the minimal sound intensity varies for sounds of different frequencies. $10^{-12} W/m^2$ is approximately the just-perceptible intensity of sound at a $1000Hz$ frequency, which belongs to the

span of frequencies human beings are most sensitive to. A normal conversation generates a sound level that is about 10^6 times larger than the just-perceptible sound intensity. Therefore, for a user to obtain auditory information from a computing system, the sound intensity should be no less than $10^{-6}W/m^2$. For the values of Ω and D given above, this results in an acoustic energy requirement of about $10^{-8}J$. Moreover, the above thresholds assume no noise (just-perceptible intensity) or relatively low noise (conversational intensity). When ambient noise increases, the output sound level has to increase accordingly, according to Webber's Law [63].

3.1.3 Power reduction techniques

Based on the above discussion, we can formulate the power requirement of a visual/auditory output as follows

$$P \propto \frac{\Omega \cdot D^2}{\eta(\lambda) \cdot V(\lambda)} \tag{3.1}$$

where $\eta(\lambda)$ is the conversion efficiency from electrical power to light/sound radiant power for wavelength λ, and $V(\lambda)$ the relative human sensitivity factor. Most display research efforts have been devoted to improving $\eta(\lambda)$ by adopting new display devices. For organic light-emitting devices (OLEDs), the best $\eta(\lambda)$ so far is $70 lumen/W$ for $\lambda = 550nm$ [58]. This is about 10-fold smaller than the theoretical $683 lumen/W$ upper limit [22].

Reducing the viewing/hearing distance D seems to be the most effective way to reduce output power requirement. Unfortunately, it poses a practical problem for visual output since it requires changes to the way a display is used. Moreover, reducing D may also have an impact on other display parameters such as pixel size and aperture (the ratio of the effective area to display area). A head-mounted display is a successful example where a reduced D is used. However, it is promising only for limited scenarios such as military and virtual reality applications at this moment. Unlike head-mounted displays, their auditory counterparts, earphones, are quite popular. Due to their extremely small D and Ω, earphones are much more power-efficient than loudspeakers, as we will see in Chapter 5.

Moreover, many applications do not need a large viewing/hearing angle. The viewing/hearing

angle can be controlled to reduce output power consumption too. Another hint from Equation (3.1) is that choosing the colors/sounds with a higher human sensitivity, thus higher $V(\lambda)$, will also reduce power. Human vision sensitivities to different colors differ by several orders of magnitude. However, user experience with colors is quite complicated since color contrast and aesthetics also matter.

3.2 Input/output speed

The energy consumption per task depends not only on power consumption but also on the task duration, or speed. We next characterize input/output speeds for human-computer interaction, which will be used to compare the energy efficiency of different interfacing technologies in Chapter 5. This subsection draws upon many previous surveys.

Speaking/listening/reading speeds: 150 words per minute (wpm) is regarded as normal for conversational English for both speaking and listening. When speaking to computers, users tend to be slower at about $100wpm$ [112]. Also, users can listen to compressed speech at about $210wpm$ [160]. Such speaking and listening rates set limits to the energy efficiency of speech-based interfaces, as shown in Chapter 5. Furthermore, when speech-recognition errors have to be corrected, the speaking rate is reduced drastically to as low as $25wpm$ [112]. For reading printed English text, 250 to $300wpm$ is considered typical [26]

Text entry: Text entry on mobile systems is well-known to be much slower than on PCs with a full-size QWERTY keyboard. Table 3.1 summarizes results from the literature about input speeds for popular text entry methods available on commercial mobile systems, such as HP iPAQ and Sharp Zaurus, which are studied in this work. "Typical speed" refers to the raw speed regardless of accuracy while "Corrected speed" refers to real speed when error correction is taken into consideration. Note that handwriting speed is for hand-printing, which serves as an upper bound for the input speed for any handwriting recognition-based text entry. The corrected word per minute ($cwpm$) for handwriting recognition is around 7 [35]. We assume that the error rate is low for hardware mini-keyboard thumbing, i.e., typing with two thumbs, and error correction is fast, as assumed for the virtual keyboard in [35].

Table 3.1: Typical text-entry speeds for different methods

Method	Typical speed (wpm)	Corrected speed ($cwpm$)
Hardware mini-keyboard thumbing	23 [197]	22
Virtual keyboard with stylus	13 [186]	12 [35]
Handwriting	15 [139]	7 [35]

Stylus/touch-screen: For GUI-based human-computer interaction, the speed is usually dependent on how fast the user can respond to the GUI. The user has to perceive the computer output through a *perceptual process*, make a decision through a *cognitive process*, and then carry out the decision through a *motor process* to respond to the computer. In [233], we characterized the user delays and investigated how they could be predicted for aggressive power management. As we are more interested in typical delays for energy-efficiency evaluation, we assume that a 500 to $1000ms$ user delay is typical for GUI operations on mobile systems such as handhelds.

3.3 Chapter summary

This chapter examined how human factors impose limits on energy efficiency with regard to interface power/energy consumption and user productivity (speed). It highlighted the importance of user interfaces and human factors, as compared to computing, and provided theoretical foundations for improving user interfaces for better energy efficiency. It also offered the theoretical minimum power/energy requirements for interfacing. Although such requirements are orders of magnitude larger than those for computing, they are still far beyond the reach of state-of-the-art user interfaces as we will see in the following two chapters.

Chapter 4

Energy Consumption of Graphical User Interfaces

In the previous chapter, we theoretically analyzed how human factors impose limits on the energy efficiency of a mobile system. In this chapter and next, we characterize the energy consumption of state-of-the-art user interfaces on mobile systems. Such a characterization is the first step toward energy-efficient user interface design.

Because GUIs have become the basic software mechanism for human-computer interaction, in this chapter, we investigate their energy consumption on three popular mobile GUI platforms. We first offer background information on mobile GUI platforms in Section 4.1. Then we analyze how energy is consumed by a GUI in Section 4.2, and describe the experimental setup and benchmarks in Section 4.3. Based on the experimental results presented in Section 4.4, we offer insights for energy-efficient GUI design in Section 4.5. We summarize the chapter in Section 4.6.

4.1 Background

In this section, we first discuss the relevant features of mobile software, and then provide information on mobile GUI platforms and development toolkits studied in this chapter.

4.1.1 Mobile software

Most of the software loaded on mobile systems is interactive (some exceptions are video/audio players, which are CPU-intensive). Such a software has two prominent properties. First, the execution time usually does not depend on the CPU speed, but on the user speed. Second, most system resources are dedicated to human-computer interaction.

User interfaces consisted of an average of 48% of the application code even a decade ago [154]. The use of modern GUIs will only increase their share of the application code and resource usage. As a result, it is important to optimize GUIs for energy savings. A GUI is responsible for interacting with users. First, it presents information to users graphically, usually through GUI windows such as buttons, menus, message boxes, and text windows. Second, it takes inputs from a user in the form of a user responding to GUI windows. Most GUI platforms are extended to include many non-user interface system functionalities, such as file and network operations, typically by wrapping corresponding system calls into a GUI API. In this chapter, we are only concerned with those parts that present and receive user input [153].

4.1.2 Mobile GUI platforms

Almost no GUI application is programmed to directly manipulate display devices. Different APIs are used to accelerate GUI development and improve hardware independence. Such APIs are called GUI platforms. In this chapter, we study three of the most popular GUI platforms on mobile systems, i.e., Qt, Microsoft Windows (or Windows), and X Window system. Since the nomenclatures used by these GUI platforms are quite different, Table 4.1 summarizes the different terms.

X Window system: The C-based X Window system [218] is almost ubiquitous on computers running under Unix-like operating systems (OSs). The X Window system has a client-server architecture, in which a single X server serves requests from different GUI applications (called X clients) through inter-process communication. Each application registers the types of events it wishes to handle with the X server, and also registers an event processing routine with each registered event type. When events associated with a window occur, the X server only sends events of registered

Table 4.1: GUI nomenclatures

	Qt	Windows	X/GTK
Event	Signal/ Event	Message	Signal/ Event
Window	Widget	Control/ Window	Widget
Event processing routine	Slot	Callback	Callback

types to the window, which in turn calls the corresponding processing routine. The X server handles events of unregistered types as the default. Wrapper toolkits are typically used to facilitate GUI development. GTK [67], one of the most popular toolkits, is used in this chapter. An embedded port of GTK, called GPE [77], is actively under development. In this study, we use the X Window system that comes with the Familiar project [204] and GTK-related libraries from the public Skiff cluster [205]. We use X/GTK to refer to the X Window system and GTK.

Qt: The Qt platform is a C++ based GUI API. It handles events through class member slots and signals. While Qt works with multiple OSs, its embedded port, Qt/Embedded [174], currently only works under Linux. Unlike Qt for PCs, Qt/Embedded applications directly work on the kernel framebuffer without an X server. The absence of an X server reduces its memory requirement significantly. One can also expect an improvement in its performance and energy efficiency. Qtopia, an application environment, has been developed using Qt/Embedded. There is also an open-source fork of Qtopia called Opie [161]. In this study, we use the Qt/Embedded system shipped with a Sharp Zaurus handheld. In the following discussions, Qt is used to refer to Qt/Embedded.

Windows: Unlike Unix/Linux systems, Microsoft Windows GUI is integrated with the Windows OS. Such an integration may offer Windows benefits in terms of GUI energy efficiency. Every Windows window has an event handler. All events generated within a window are passed to its event handler through an event loop. There are multiple ways to develop GUI applications for Windows, e.g., using Win32 API, MFC, ATL, or Visual Basic. Unlike the X Window system, a significant number of Windows developers use the Win32 API directly to write Windows applications. There-

fore, the Win32 API is adopted in this chapter for energy characterization.

Other GUI platforms: We do not address other mobile GUI platforms in this first study. One of the better known is the Palm GUI widely used with Palm OS [163] powered handhelds. No Palm handheld uses an Intel StrongARM processor yet, which would make a fair comparison difficult. Java and Visual Basic are also popular for GUI development. However, their well-known performance disadvantages compared to C/C++ are also translated into energy disadvantages, although they may have advantages in other aspects. Other GUI platforms/toolkits for embedded Linux are described in a recent article [203].

4.2 GUI energy consumption

Before energy characterization, it is worthwhile to analyze how energy is consumed by a GUI. In this section, we will do so from three perspectives: hardware, software and user.

Hardware: In the mobile systems we study, the Intel StrongARM SA-1110 SoC is used. It has an integrated LCD controller (LCDC). A framebuffer is implemented in off-chip memory (main memory) to store pixel data for a full screen. Whenever there is a screen change, the processor generates new data for the changing screen pixels and stores them into the framebuffer. This implies more energy consumption with larger temporal changes in the screen. Meanwhile, to maintain a screen on the LCD, the LCDC must sequentially read screen data from the framebuffer and refresh the LCD pixels even when there is no screen change. This in turn implies more energy consumption with larger spatial changes in the screen. The on-chip system bus and off-chip data buses also consume energy for data transfers.

The display itself consists of several parts: LCD power circuitry, a front light, and an LCD. The LCDs used in the systems we studied are color active thin film transistor (TFT) LCDs. In such LCDs, each pixel has three components: R, G and B, signifying red, green and blue, respectively. Liquid crystals for each component are independently oriented by two polarizers, which are connected to a storage capacitor. The capacitor is in turn charged and discharged through a TFT to accommodate screen changes. It must be refreshed at a high rate to maintain an appropriate voltage across the polarizers so that the corresponding liquid crystals remain properly oriented. The

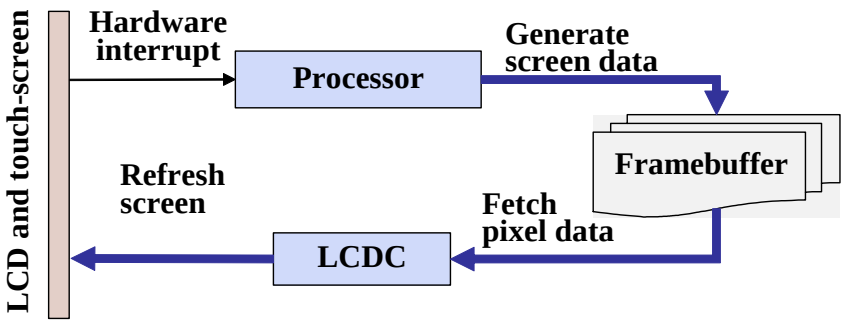


Figure 4.1: A hardware perspective of GUI energy consumption.

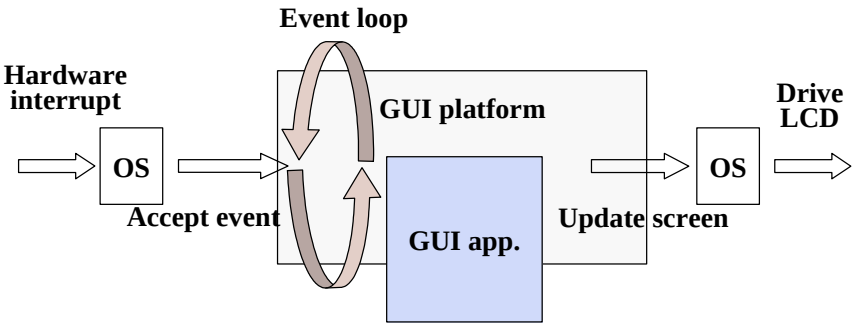


Figure 4.2: A software perspective of GUI energy consumption.

hardware perspective is summarized in Figure 4.1.

Software: A GUI platform is highly OS-dependent. It is impossible to compare GUI platforms without taking the OS into account. The following software processes are involved in GUI usage. First, the OS handles hardware interrupts generated by a user. It then produces events for the GUI platform. The latter delivers events to the GUI application, which catches events through an event loop. Thereafter, the application instructs the platform as to how the GUI should change. The platform coordinates GUIs of different applications, determines how the screen changes, generates new screen pixel data, and then calls OS services to update the screen. Interrupt handling, event processing and screen updating are the three basic steps in the above process. The software perspective is summarized in Figure 4.2.

User: From the user's perspective, a GUI consumes energy through user-GUI interaction ses-

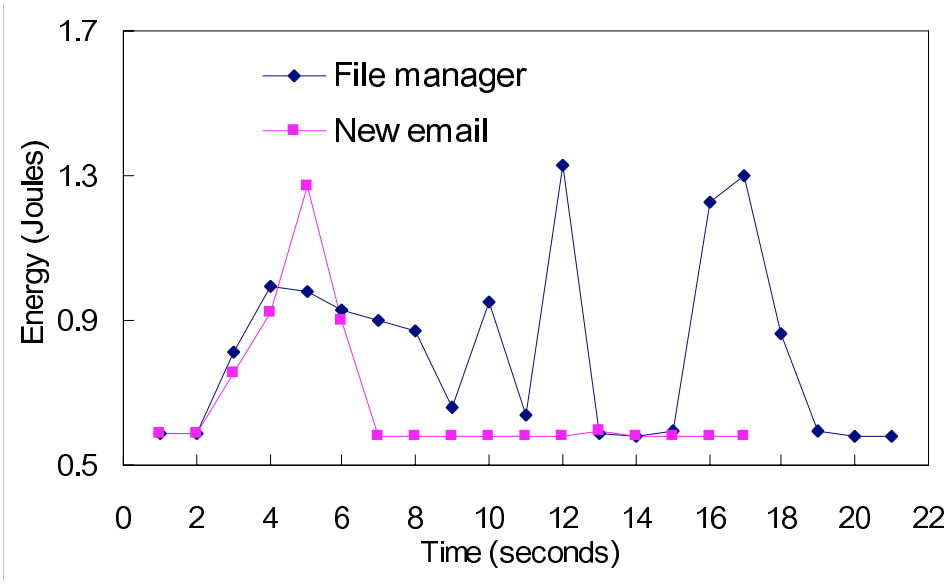


Figure 4.3: Looking for a file and creating an email.

sions, in which a user locates the application, starts it, interacts with it and finally closes it. Such a process usually consists of a series of window operations, such as creation, switching, and manipulation, with intermittent idle intervals, in which the system waits for user input.

Figure 4.3 gives the second-by-second energy consumption for two GUI sessions executed on an HP/Compaq iPAQ with Pocket PC 2002. The first session is to look for a file using the file manager. The first peak around four seconds is due to activation of the "Start" menu and stepping through its items to locate "Programs". The second peak at 10 seconds is due to activation of "Programs". The third peak at 12 seconds is due to activation of the file manager in the Programs window. The last peak is due to moving of the scroll bar in the file manager to locate the file in the file list and then closing the file manager. The other session, creating an email, is just to open the "New" menu and create a new email message. The two sessions consume 4.7 and 1.4 Joules more energy, respectively, than consumed in the corresponding idle period. Being idle for one second consumes 5.9 Joules in these measurements. Creating an email consumes much less extra energy because it requires fewer GUI manipulations. Moreover, since it takes much less time, more email sessions are possible compared to file manager sessions in a fixed amount of battery lifetime. From these two examples, it is obvious that window operations are energy-expensive and an energy-efficient GUI

Table 4.2: Hardware and software information on mobile systems

	iPAQ1	iPAQ2	Zaurus
Vendor	HP/Compaq		Sharp
Model	3870		SL5500
SoC	Intel StrongARM SA-1110 206MHz		
Storage	32MB ROM, 64MB RAM		16MB ROM, 64MB RAM
Display	240×320, 16bit color, reflective with front light		
OS	MS Pocket PC 2002	Familiar Linux 2.4.18	Embedix Linux 2.4.6
GUI	Windows	X/GTK	Qt

should reduce the number of window operations and usage time.

4.3 Experimental setup

In this section, we first provide information on the mobile systems used and the energy measurement setup, then detail the benchmarks and GUI energy characterization methodology.

4.3.1 System information

The information on the mobile systems used in this chapter is summarized in Table 4.2.

The two iPAQs are placed in the same charging cradle when energy is measured. The Zaurus, instead, is directly charged by the same AC/DC converter used for the iPAQs. Energy is measured when the batteries are fully charged. All other detachable peripherals, such as CompactFlash cards, serial and USB ports, are disconnected. Since no current GUI platform has an effect on the front light, it is turned off unless otherwise indicated.

4.3.2 Energy measurement

The measurement equipment consists of a Windows PC, hardware data acquisition card, and wire connector box. The PC is installed with *National Instrument*'s data acquisition software called $Labview$. The voltage is measured across a sense resistor connected in series with the battery to obtain the system power consumption. It is sampled at a rate of $400Hz$. $Labview$ is programmed to integrate power every second. By carefully designing the benchmarks, the energy consumption of software events of interest can be captured. The energy measurements for a given benchmark were made on mobile systems on the same day without interruption to minimize the day-to-day difference due to changes in temperature and resistance of the sense resistor.

4.3.3 Methodology and benchmarks

While the source code of Qt and X/GTK are freely available, that of Windows is not. However, the GUI APIs of all three platforms are well-documented. The GUI API is an appropriate level of abstraction for modeling GUI platforms. Based on the GUI APIs and platform architectures, we adopt a controlled black-box methodology to characterize different aspects of GUI energy consumption, as explained next.

Additional energy: To account for energy differences due to the OS and hardware, we use the concept of *additional energy*. When a software or hardware event occurs in the system within a certain time interval, the software running in the interval is called a $target$. We design the software to be exactly the same except that it does not trigger that event during the same time interval. The resultant software is called a $context$. The additional energy of the event is defined as the target energy minus the context energy. For every GUI event we characterize, a context is carefully designed. Measurement of a target and corresponding context is repeated in an interleaved way to reduce the impact of random factors.

Normalized energy: We denote the additional energy for performing certain compute-intensive jobs in each system as the *energy unit* (EU). The compute-intensive job we chose was the jpeg_fdct_islow routine in file jfdctint.c from Independent JPEG Group's implementation of JPEG, which comes with the Mibench benchmarks [142]. It performs a forward discrete cosine transform (DCT) on an

eight-by-eight block of integers. Three different sets of inputs are randomly chosen from the large image file included in MiBench. To obtain the additional energy for performing one such DCT, we repeat the DCT a total of 3×10^5 times over the set of chosen input data. This is assumed to be the target, which takes our systems about four seconds to complete. The context in this case simply involves making the system idle. The energy of every benchmark is measured with a companion measurement of the EU. In most cases, we report experimental results normalized to the EU thus obtained. This accounts for differences in the hardware and OS. The EU for the three systems we studied is between 8 and 10 μJoules.

The benefits of using the EU are as follows. Experiments were conducted on different days for different benchmarks. The absolute energy figure for an event varied slightly from day to day. However, the energy remained quite constant if normalized to the corresponding EU (within 1%). Moreover, since the EU is only dependent on the SoC and memory, the comparison of non-LCD energy consumption of different systems is fairer after normalization.

Benchmarks: We designed the benchmarks to characterize different aspects of GUI platforms including event handling, typical window operations, and window properties. Also, we characterized common window types and their related usage. Different input methods were also characterized. The benchmarks are described in Table 4.3. Unless otherwise indicated, they were coded for all three GUI platforms. The coding and compilation information is given in Table 4.4. The source code for benchmarks used in this chapter can be downloaded from [195]. Most benchmarks were coded using a similar scheme in which a timer is set to start some processing. In a target, the processing triggers the event we wish to characterize; in the corresponding context, the processing is the same except that the event is not triggered. A snapshot of the "Event loop" benchmark target coded for Windows is shown in Figure 4.4 to illustrate such a scheme. Note that its context uses the same code except that line 13 is commented out. Routines to turn the LCD off and on were inserted at the beginning and end of the benchmarks to obtain the additional energy due to the LCD.

Table 4.3: Benchmarks

Benchmark	Description
Event handling	
Event loop	Send and get an event
System event	Use stylus to tap a window which is programmed to ignore that event
Basic window operations	
Create window	Create and then destroy a window
Show window	Show and then hide a full screen window
Using windows of different types	
Menu	Show and then hide a menu window of four items
Message box	Show a dialog box with “OK” and accept user confirmation
Scroll bar	Move a scroll bar of half screen height with various speeds
Tabbed panel	Switch between two full screen tabbed panels (iPAQ1 and Zaurus only)
Window properties (iPAQ1 only)	
Size	Show and then hide menu windows of different numbers of items, normal windows of different sizes (60×80, 90×120, 120×160, 180×240, and 240×320 pixels, respectively), text windows of different text sizes
Colors	Present a full screen window of different colors
Color sequence	Show and then hide full screen windows of different colors on a black background
Color patterns	Present full screen windows of black and white checkerboard patterns with different block sizes
Different user input methods	
Virtual keyboard	Push “x” on the virtual keyboard
Hardware button	Trigger the right cursor key button
Stylus tap	Same as System event
Stylus move	Move stylus vertically along the screen for half screen height on a window which is programmed to ignore corresponding events

Table 4.4: Coding and compilation information for benchmarks

	Qt	Windows	X/GTK
Coding language	C++	C	C
IDE	N/A	eMbedded Visual C++ 3.0	N/A
Compilation setting	gcc -O2	Default	gcc -O2

```
1  #define ID_TIMER WM_USER + 300
2  static int count = 0;
3  ......
4  LRESULT CALLBACK WindowFunc(HWND hWnd, UINT message,
5                       WPARAM wParam, LPARAM lParam)
6  {
7     ......
8     switch (message)
9     {
10          case WM_TIMER:
11              if(count<TOTAL_NUM){//Repeat TOTAL_NUM times
12                  //Send a message to myself
13                  SendMessage(hWnd,WM_LBUTTONDOWN,0,0);
14                  count++;
15               } else {
16                  //Request to exit
17                  SendMessage(hWnd,WM_DESTROY,0,0);
18               }
19               break;
20          case WM_LBUTTONDOWN: //Catch the sent message
21               //Do nothing here
22               break;
23          case WM_CREATE:
24               //Set a timer
25               SetTimer(hWnd, ID_TIMER, 2000, NULL);
26               break;
27          case WM_DESTROY:
28               PostQuitMessage(0);
29               break;
30          default:
31               return DefWindowProc(hWnd, message, wParam, lParam);
32    }
33    return 0;
34 }
```

Figure 4.4: Outline of the target for benchmark “Event loop”. The context uses the same code but with line 13 commented out.

Table 4.5: System energy breakdown for mobile systems

Handheld		LCD		Front light		Others		EU
		%	EU	%	EU	%	EU	μJoule
iPAQ1	idle	9	18,800	73	147,100	18	35,300	8.0
	DCT	7		53		40	112,600	
iPAQ2	idle	14	20,600	53	77,700	33	48,200	9.8
	DCT	9		34		57	129,100	
Zaurus	idle	11	25,200	80	180,000	9	19,000	9.3
	DCT	8		59		33	99,300	

4.4 GUI energy characterization

In this section, we present experimental results for the three mobile systems, and analyze them from the perspectives of software, hardware, and user.

Energy breakdown: Table 4.5 gives the energy breakdown by hardware for the systems targeted when they are idle and while performing the aforementioned DCT computation. It also gives the one-second energy consumption in EUs for different components. The LCD energy is obtained by comparing the system energy before and after the LCD is turned off. Front-light energy is obtained in the same way. "Others" refers to the system energy minus the LCD and front-light energy. It includes the energy consumed by all other hardware. The table shows that the front light and the TFT LCD consume a large fraction of system energy. The percentages are larger than those reported for notebook computers [31], in which a hard-disk, system-on-board and more powerful processor are used instead of Flash memory and SoCs.

With the front light turned off, the TFT LCD consumes from 14% to 55% of the system energy, depending on how busy the CPU is. However, it should be noted that during interactive application usage, the CPU is idle most of the time.

Event handling and basic window operations: Table 4.6 presents the additional energy in EUs for GUI event handling and basic window operations. “Event loop” shows the additional energy for an event to go through the event loop. It is very energy-efficient compared to “System event,” which includes additional energy for hardware interrupt, OS and platform event processing. Windows outperforms Qt and X/GTK significantly, which may be attributed to tighter integration of its GUI platform and OS.

“Create window” shows the additional energy for a GUI platform to claim and relinquish resources for a window according to the request from the applicaion. “Show window” shows the additional energy for the GUI platform to show and hide a full screen window according to the request from the application. The background window is identical to the one shown so that no additional energy is incurred due to framebuffer updating, LCDC or LCD. The additional energy can only be attributed to changes in the internal data of the GUI platform as in the cases of “Event loop” and “Create window”. While Qt and Windows are relatively close, X/GTK performs significantly worse in “Show window.” This hints at the overhead of an X server.

Window types: Table 4.6 also shows the additional energy required for using different types of windows and showing an eighty-letter text. It is obvious that using different window types consumes quite different amounts of energy even when the window sizes and colors are similar, as in a message box and a four-item menu window. Different window types necessitate different user interactions too, which further differentiate their energy consumption. There are several observations worth noting, as discussed next.

First, the same interactive function may be implemented using different window types with different energy efficiencies. For instance, both tabbed panels and a scroll bar can be used to browse a long list. Their energy consumption differs drastically since a scroll bar requires many more screen updates than tabbed panels. Second, “Message box” of Windows outperforms others simply because it allows a “Message box” to be much simpler than Qt or X/GTK does. For example, “Message box” of Qt has 3D effects and must contain at least one button. This indicates that an inflexibility in the GUI platform can cause extra energy consumption. Finally, Windows performs much worse for “Menu,” although it does well in other window types, because it animates menu

Table 4.6: Energy characterization in EUs for different GUI platforms

	Qt	Windows	X/GTK
Event handling			
Event loop	27	11	40
System event	1,100	300	900
Basic window operations			
Create window	1,900	2,600	1,000
Show window	9,900	7,900	18,000
Using windows of different types			
Menu	12,800	15,400	6,100
Message box	14,400	6,000	13,300
Scroll bar	33,400∼78,000	20,000∼59,000	38,800∼84,000
Tabbed panel	10,300	12,000	-
Draw text	-	1,600	-

windows. When a menu is tapped, the menu window gradually, but quickly, drops out. Such an animation or continuous screen change requires many more processor cycles, framebuffer updates and screen refreshes, thus leading to much more additional energy. Using a scroll bar to browse a window also requires continuous screen changes. For energy efficiency, such continuous screen changes should be avoided, at the expense of a slight sacrifice in GUI aesthetics.

Size: We have experimented with different sizes for text, menus and windows. Figures 4.5 to 4.7 show the additional energy required. They also show the equation for the best-fit line, obtained through linear regression. The relationship between the size and additional energy is approximately linear. Each letter consumes about 12 EUs. Each new menu item consumes about 220 EUs. These data suggest that GUI designs should be economical to be energy-efficient. Moreover, the huge

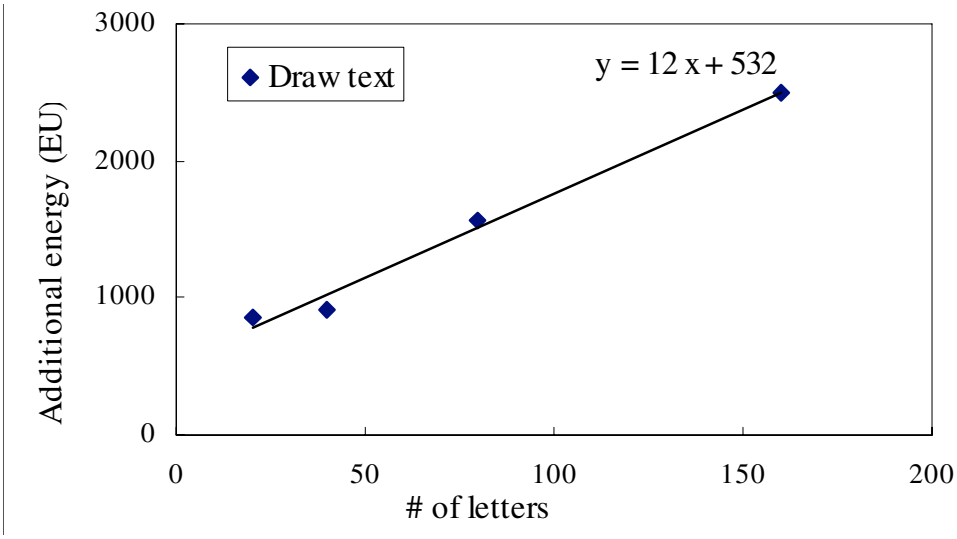


Figure 4.5: Additional energy for showing texts of different sizes.

constants in the linear regression equations also imply that if a user needs to view a large number of items, putting them in as few windows as possible saves significant energy.

Color: Color affects the additional energy consumption of a GUI in several ways. Our first experiment measures the energy consumption of iPAQ1 when the CPU is idle for one second with screens of different colors. Since each pixel consists of three color components, R, G, and B, we perform measurement for colors containing different combinations of these three components. We also perform measurements with the LCD turned off to obtain the energy consumption of the LCD only. Table 4.7 summarizes the results under energy consumed by the LCD and the non-LCD energy. It also shows the percentage energy increase compared with pure white. R, G, B, and RG refer to red, green, blue, and yellow, respectively. They have R, G, B, and R and G component(s) deactivated, respectively. “Grey” refers to the color obtained when the corresponding originally activated component(s) is (are) half-activated. For example, when all three components are fully activated, the color is “Full” black. When they are half-activated, the color is “Grey” black. The energy difference disappears after the LCD is turned off, which demonstrates that it is the LCD that makes the difference.

There are two observations one can make. First, the more color components activated, the more the energy consumption. In a reflective TFT LCD, when one color component is deactivated, the corresponding liquid crystals are fully unpolarized, and there is no need to repeatedly charge the

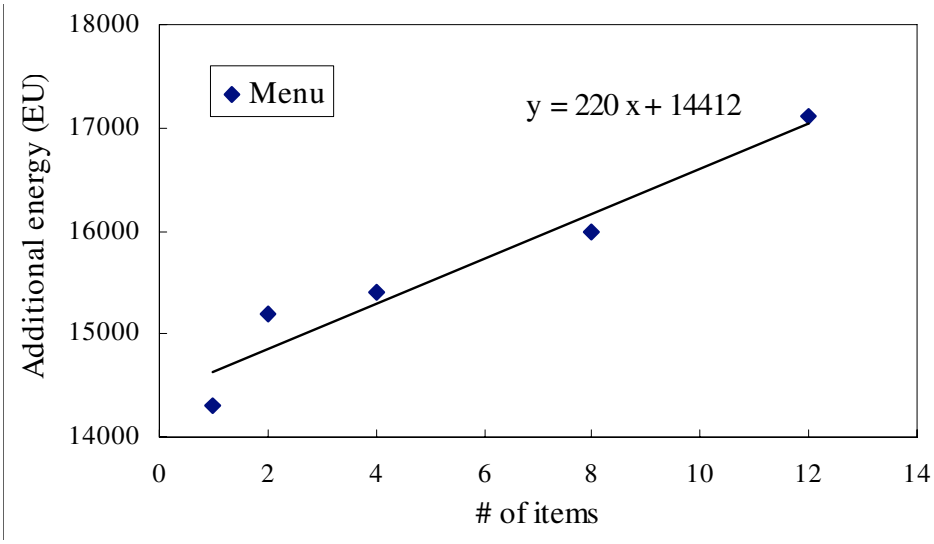


Figure 4.6: Additional energy for showing menu windows containing different numbers of items.

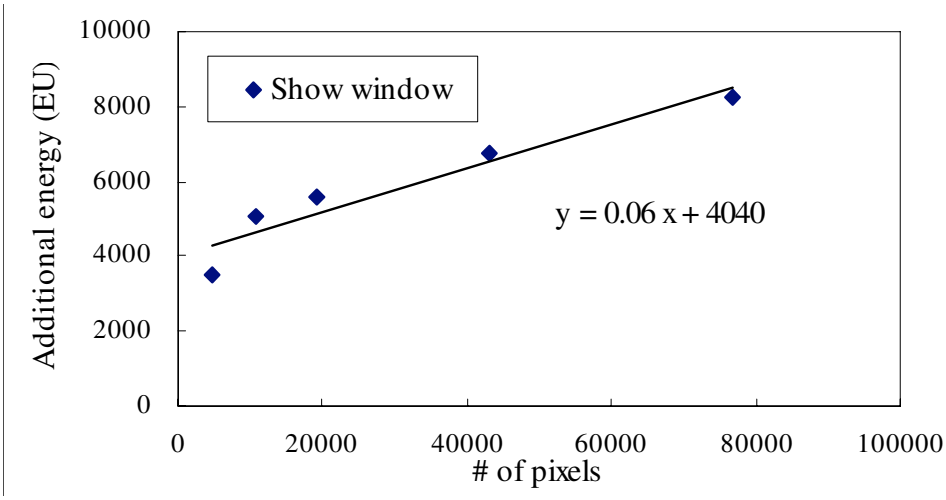


Figure 4.7: Additional energy for showing windows of different sizes.

Table 4.7: Energy breakdown for presenting screens of different colors

Color		Non-LCD (EU)	LCD (EU)	% Increase
Black	Full	36,100	18,700	3.3
	Grey	36,100	19,100	5.5
Red	Full	36,100	18,500	2.2
	Grey	36,100	18,700	3.3
Green	Full	36,100	18,500	2.2
	Grey	36,100	18,700	3.3
Blue	Full	36,100	18,600	2.8
	Grey	36,100	18,700	3.3
Yellow (RG)		36,100	18,300	1.1
White		36,100	18,100	0

polarizers. For instance, “White” has all three color components deactivated and consumes the least energy, while “Black” has all three activated and consumes more energy. This observation is the same as that made in [29] for a transmissive TFT LCD. More interestingly, the second observation is that a half-activated component consumes more energy than an activated component, as the “Grey” ones consume more energy than their “Full” counterparts. As mentioned in Section 4.2, each color component has a TFT to charge a storage capacitor, which maintains the appropriate voltage between the polarizers. When a component is half-activated, the capacitor may put the TFT into a state that draws a higher amount of current, which contributes to higher energy consumption. It must also be mentioned that the relation between pixel power consumption and storage capacitor voltage can be different for different LCD technologies. For example, we find that power consumption for “Grey” is between that for “Full” and “Black” for the Zaurus LCD. Cheng and Pedram [28]

Table 4.8: Energy for different colors for the QPE theme on Zaurus

Theme color	Energy (Joule)	Over Bright(%)
Bright	0.416	0
Purple	0.417	0.4
Desert	0.422	1.5
Grey	0.423	1.6

also made a similar finding for a transmissive LCD from Philips. Nevertheless, the impact of color on LCD power consumption is obvious.

On Zaurus, users can choose different colors for GUI themes. Table 4.8 gives the system energy for presenting the “Application” window with different colors for the QPE theme for one second. It confirms the observation made in the iPAQ1 experiment. It also shows the percentage system energy increase when compared with that of the “Bright” theme color.

Color sequence: As the analysis in Section 4.2 reveals, temporal changes in the screen cost energy. When the screen changes color, the CPU consumes additional energy to generate data for the framebuffer, the framebuffer has to be then updated, and corresponding liquid crystals have to change orientation. The second experiment was designed to measure the energy of iPAQ1 for changing the screen from black to different colors. To eliminate the additional energy due to the LCD presenting different colors, it is turned off. Therefore, the experiment mainly accounts for the additional energy due to the first two of the three processes mentioned above. The data are presented in Table 4.9. It demonstrates the more the color changes, in terms of (R,G,B) components, the larger the additional energy consumption. This implies that a GUI with a constant color theme will be more energy-efficient than one that often changes color.

Color patterns: As also evident through the analysis in Section 4.2, spatial changes within a screen also impact energy consumption. We call how colors are distributed on the screen its *color pattern*. The color pattern determines the spatial changes within the screen. Even if the percentage

Table 4.9: Additional energy for showing and hiding windows of different colors on a black background

New color	White	Yellow (RG)	Red (R)	Grey	Black
Add. energy (EU)	8,000	7,900	7,500	7,700	7,400

of pixels of each color remains constant (then the LCD consumes constant energy), different pixel arrangements can introduce different switching activities in the hardware including the LCDC, system bus, and external bus, because data for the screen have to be constantly transferred from the framebuffer sitting in the off-chip memory to the LCDC, and then to the LCD for refreshing the storage capacitors. We measure the energy consumption of the system when the system is idle with full-screen windows of checkerboard patterns on iPAQ1. A checkerboard pattern consists of alternating white and black blocks. For each pattern, the white and black blocks each take up half of the screen pixels so that the LCD energy consumption does not vary. However, as expected, the energy consumption increases when block size decreases, which leads to more spatial changes within the screen. Table 4.10 shows the energy consumption for one second and also gives the percentage energy increase compared with that of presenting a fully white screen. It also shows the system energy for presenting the starting home screen for Pocket PC 2002. The energy difference in Table 4.10 is due to the showing of the screen only. The screen with smaller blocks takes more CPU time and energy to generate the screen data, which is a separate issue from what we are concerned with here. The above results imply that a plain GUI is more energy-efficient than fancy ones.

Input method: From the user's perspective, a user interacts with GUIs through different input methods, and uses a certain input method to interact with a certain type of window. Table 4.11 provides the energy consumption for different input methods on the targeted mobile systems. It also shows the number of stylus taps an input method is equal to (denoted by #) with regard to energy consumption. The stylus tap is the most commonly used input method. Hardware buttons are used to trigger the most-used applications. Virtual keyboard is necessary for text input, although Zaurus also comes with a mini hardware keyboard that we do not characterize in this chapter. Stylus move

Table 4.10: Different color patterns on iPAQ1

Pattern	Energy (Joule)	Over white (%)
Full white	0.575	0
Full black	0.581	1.0
120×160 block	0.577	0.3
30×40 block	0.584	1.6
12×16 block	0.588	2.3
3×4 block	0.598	4.0
MS home	0.590	2.6

is typically used to move a window like a scroll bar.

A stylus move is very expensive in all three systems. Moreover, it is usually associated with continuous screen changes such as window moving and resizing, for which the energy cost is significant. Moreover, a virtual keyboard is very expensive on iPAQ1 and Zaurus because they have the auto-completion feature in order to accelerate user input. Since reducing usage time saves a significant amount of energy, auto-completion is generally more energy-efficient. This issue is further addressed in [213]. Moreover, inputting text is much slower using a virtual keyboard than a real keyboard. Thus, it increases usage time significantly on mobile systems, and leads to more energy consumption from the user's perspective. For energy efficiency, stylus move and text input should be minimized.

GUI platform comparison: In general, all three GUI platforms characterized in this chapter perform similarly with regard to their energy usage. All are descendants of GUI platforms used in non-energy critical computers. Most likely, none of them is designed with energy efficiency as a goal. Each contains areas for improvement. Windows performs better in event and interrupt handling due to its tighter integration with the OS, but suffers due to its PC lineage. X/GTK is

Table 4.11: Additional energy for different input methods

Input method	Qt		Windows		X/GTK	
	EU	#	EU	#	EU	#
Stylus tap	1,100	1	300	1	900	1
Hardware button	1,400	1.3	560	1.9	1,300	1.4
Virtual keyboard	4,000	3.6	4,700	15.7	1,200	1.3
Stylus move	∼13,500	∼12.3	∼5,700	∼19.0	∼3,000	∼3.3

better in providing most implementation flexibility for designing an energy-efficient GUI, but suffers from using an X server and lack of enough features. Qt offers more features for a fancy user interface, but suffers from limited flexibility for making a GUI simpler. Moreover, since each of them only works with a unique OS and requires a different license, a platform choice is further complicated. Nevertheless, the energy characterization of GUI platforms presented here should help GUI designers make a better decision and help GUI platform/toolkit developers improve their work with regard to energy consumption.

4.5 GUI design for energy efficiency

As the results presented above show, the energy impact of GUIs is significant. It is extremely important to optimize them when energy consumption is of concern. Although GUI design is an established discipline in the community of human-computer interaction, no previous research takes energy efficiency into consideration. Based on the GUI energy characterization presented in this chapter, we next offer some first insights into energy-efficient GUI design, and compare them with the traditional do's and don'ts for GUI design [109] when possible.

Improve user productivity: Mobile systems spend most of their time and energy during idle periods in human-computer interaction. The energy consumed by most GUI operations, as charac-

terized in this chapter, will appear insignificant when compared with that consumed in idle periods. Hence, reducing the task time is the most effective way of energy reduction, and an energy-efficient GUI should be designed for maximum user productivity. Fortunately, user productivity has been one of the most important concerns of conventional GUI design. For example, fast access to functionalities is regarded as important by traditional GUI design [109] (pp. 62-78). With a direct impact on energy efficiency, user productivity becomes even more important for mobile systems.

Minimize screen changes: There are several ways for minimizing screen changes, some of which agree with the traditional methods for GUI design to preserve display inertia. For example, GUI changes may be cached. That is, consecutive changes taking place within a short time interval can be presented as one single change without affecting user interaction. This also means that continuous screen changes as animation and window scrolling should be avoided, which disagrees with the traditional method for providing visual continuity [109] (p. 282).

Avoid or minimize text input: Traditional wisdom [109] (pp. 121-128) makes this recommendation to minimize user switching between a mouse and keyboard. This is even more important from the energy point of view for mobile systems since text input is much slower for them. For example, if the range of inputs is known beforehand, a list can be supplied to ask users to choose, instead of type in, text.

Reduce redundancy: This both disagrees and agrees with traditional wisdom. Traditional wisdom asks for an animated progress indicator to improve software responsiveness while a simple busy indicator may be more energy-efficient. Traditional wisdom also asks for consistent menus which keep useless menu items on the menu window, but deactivate them, in order to improve user friendliness [109] (p. 62). However, such items may be removed to save energy once a user has become familiar with the software. To summarize, features that do not improve user productivity should be avoided. This does agree with traditional wisdom on task queue optimization for ignoring outdated user requests [109] (p. 397).

Do something while waiting for user input: Since being idle consumes a lot of power, a more aggressive way to improve user productivity is to speculate on what may be the user input and get the result ready before the next input is provided. For example, auto-completion in the virtual

keyboard is one way of speculating. Moreover, tasks can be reordered to first present those windows on the screen that ask for user input, and then go ahead with processing of other tasks. This idea coincides with the traditional wisdom of dynamic time management [109]. However, it is used for a different purpose here.

Style and color: First, being economical and terse is important, as we have shown that the window size and text size do matter in terms of energy. Second, choosing colors that consume less power on the display is also important. For example, colors with fewer color components consume less power for TFT LCDs, as shown in this chapter. Moreover, color changes from window to window and fine color patterns or highly decorated windows should also be avoided, and plain windows should be preferred.

Make GUI energy-aware: While energy efficiency refers to energy consumption per task, *energy awareness* refers to the capability of trading other aspects of a task for energy savings. Many insights offered above save energy with sacrifice in another aspect. For example, not showing progress bars may be somewhat awkward; inconsistent menus may be slightly confusing; a plain theme may render software less attractive. Moreover, new display technologies [14, 29, 61, 167] allow tradeoffs between GUI aesthetics and energy savings. Instead of using an energy-efficient GUI all the time, a mobile system needs an energy-aware GUI that adapts to energy availability for the best tradeoff.

4.6 Chapter summary

This chapter presented the first study to characterize the energy consumption of GUIs implemented on three popular mobile GUI platforms. It analyzed the GUI energy consumption from the perspectives of hardware, software, and user, demonstrating that a GUI has a significant impact on energy consumption. It highlighted the energy impact of interfacing, as compared to "computing." Based on the characterization, it offered insights for improving GUIs for energy savings, and provided a solid foundation for further research on energy-aware and energy-efficient GUI design [213].

Chapter 5

Energy Efficiency of Mobile User Interfaces

In Chapter 4, we presented an energy characterization of mobile GUIs. Without quantitatively considering user productivity, however, we were unable to truly evaluate the energy efficiency of a GUI. In this chapter, we first extend our energy characterization to other state-of-the-art interfaces, and then evaluate their energy efficiency based on user productivity information. The chapter is organized as follows. We first describe the experimental setup in Section 5.1. Then we present an energy characterization of visual and auditory interfaces, and manual input techniques in Sections 5.2, 5.3 and 5.4, respectively. We present a comparative study of them in Section 5.5. After that, we present observations for techniques presented in Chapter 4 and this chapter in Section 5.6. We summarize the chapter in Section 5.7.

5.1 Characterization setup

Table 5.1 provides information on system settings and input methods for the two mobile systems characterized in this chapter. Both are new models of the iPAQ and Zaurus used in the previous chapter. iPAQ is also equipped with Bluetooth. Note that several different handwriting recognition schemes are available on both computers. The user can input text letter-by-letter using letter or

Table 5.1: System information for iPAQ and Zaurus

	iPAQ	Zaurus
Model	HP iPAQ 4350	Sharp SL5600
SoC	Intel XScale $400MHz$	
Storage	$32MB$ ROM, $64MB$ RAM	$16MB$ ROM, $64MB$ RAM
Display	240×320, 16-bit color	
	Transflective/back light	Reflective/front light
OS	MS Pocket PC 2003	Embedix Plus PDA 2.0 (Linux 2.4.18)
Battery	$1560mAh/3.7V$	$1700mAh/3.7V$
Text entry	Touch-screen with stylus	
	Hardware mini-keyboard (QWERTY)	
	Virtual keyboard (QWERTY)	
	Handwriting recognition	
Image/Video	N/A	CF digital camera
Audio	Integrated mic., speaker & headphone jack	
Speech recog.	Voice Command [145]	N/A

block recognition on both systems. The user can also input a group of letters using Microsoft *Transcriber* [144] on the iPAQ.

Power measurements: Power measurements are obtained by measuring the voltage drop across a $100m\Omega$ sense resistor in series with the $5V$ power supply cord. The measurement system consists of a Windows XP PC with a GPIB card and an HP Agilent 34401A digital multimeter. A program, developed with Visual C++, runs on the PC and controls the digital multimeter to measure the voltage value. The value is sampled about 200 times per second.

Basic power breakdown: We first characterize the power consumption due to hardware activities initiated by user interaction. We use the power consumption of idle systems (in the IDLE mode) with the display off as the baseline, and present the power consumption of additional hardware activities as additional power consumption relative to the baseline. The additional power/energy consumption of an event is obtained through two measurements, as described in Section 4.3: one for the system power/energy consumption during the period an event of interest occurs; the other for the system power/energy consumption during the same period when the event does not occur. For example, the additional power consumption of the LCD is obtained by subtracting the system power when the system is idle and the LCD is off from that when the system is idle and the LCD is on. The power characterization results are presented in Figure 5.1. In this figure, "BT Trans." refers to Bluetooth transmitting data at 9.6 Kbps; "BT Paging" refers to Bluetooth seeking a connection with another device, and "Comp." refers to measurements when the system is repeatedly performing the DCT described in Section 4.3.

5.2 Visual interfaces

We first examine visual interfaces.

Graphical user interface: In Chapter 4, we presented a comprehensive analysis of the energy consumption required for GUI manipulations. In [233], we showed, however, that most of the system energy is consumed when the system waits for the next user input. If we ignore the additional energy consumed by the system to generate a GUI response, GUI manipulation-based interfaces basically consume energy through a static display and an idle system. As pointed out in Chapter 4,

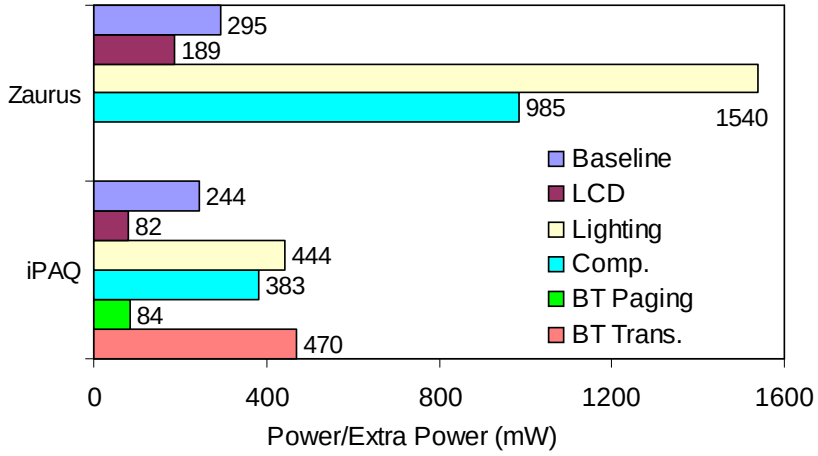


Figure 5.1: Baseline power and additional hardware power consumption

the most effective system energy reduction strategy is to improve user productivity so that more tasks can be accomplished given the same battery lifetime. In Section 5.5, the energy efficiency of GUIs is compared with other interfacing technologies based on the length of the corresponding GUI operation.

Visual input: Gesture recognition and lip-reading have been proposed as possible techniques for multi-modal human-computer interaction. Both require video or image input. We used a CF digital camera card on Zaurus to obtain its power cost. When the camera is turned on with a 480×320 resolution and faces a static object, the system consumes about $1.35W$. When the object moves, the power consumption increases slightly to about $1.36W$. This is close to the power consumption when the user is preparing for a shot, $e.g.$, adjusting the focus and view. Also, it takes about $0.33J$ to capture a 480×320 picture. Since a user usually takes more than a few seconds to prepare a shot, it is obvious that it is more important to reduce the user's preparation time and system power consumption during that time than reduce these for actually capturing the picture.

5.3 Auditory interfaces

We next examine the auditory interfaces available on iPAQ and Zaurus.

5.3.1 Direct recording and playback

An auditory signal can be directly recorded and played back for interfacing purposes. Direct recording is often used for note-taking and direct playback for short sound responses from the computers such as warnings and notifications. If there are too many sound responses to be feasible for direct playback, speech synthesis is required.

Direct recording: iPAQ provides a hardware button to start recording ($11KHz$ 16-bit Mono), which is very useful for audio note-taking. The recording consumes $525mW$. The additional power consumption is thus $199mW$. Zaurus draws about $198mW$ additional power consumption when recording ($16KHz$ 16-bit Mono).

Direct playback: A WAV sound clip ($32KHz$ 16-bit mono) was played on both iPAQ and Zaurus. To separate the power consumption of the speaker subsystem, the clip was played at different volumes. Table 5.2 shows the power consumption under various scenarios. "Half" volume assumes that the volume controller is set at the half mark on each system. In this scenario, the clip is not comfortably enjoyable on either system, even in a quiet office environment, if the system is about two feet from the user head. All the system power numbers include that consumed by the LCD.

The additional power consumption of the speaker subsystem is obtained by comparing the system power consumption before and after the system is muted. This has a significant impact on system power efficiency if the auditory output is used. Notably, using an earphone instead of the built-in loudspeaker reduces power by more than $300mW$ and $410mW$ for iPAQ and Zaurus, respectively. The data also indicate that using a simpler audio format (WAV as opposed to MP3) reduces power consumption at the cost of increasing the storage requirement. Since the additional power consumption of the speaker subsystem for playing MP3 is similar to that for playing WAV, the power consumption for playing MP3 at different volumes is not presented.

5.3.2 Speech recognition and synthesis

We next examine the Microsoft Voice Command [145] on iPAQ to obtain its power for a speech recognition-based interface. Voice Command is similar to the MiPad Tap & Talk system [121] except that synthesized speech is used as feedback to the user. Not having a detailed knowledge of

Table 5.2: Power consumption for different auditory outputs

Format	Volume	iPAQ (mW)			Zaurus (mW)		
		System	Extra	Speaker	System	Extra	Speaker
WAV	Max.	747	420	367	1,030	546	422
	Half	552	232	172	637	153	29
	Muted	380	53	0	608	124	0
	Earphone Max.	445	118	65	619	135	11
MP3	Earphone Max.	476	149	N/A	632	148	N/A

its implementation, we adopted a black-box approach. We recorded both the power trace and the audio input/output and then aligned them to divide the power trace into meaningful segments. We fed different inputs to Voice Command to elicit certain behaviors from it.

Speech acquisition without speech being detected: We first evaluated Voice Command under no sound. Hence, the speech detection module does not detect any speech. A reasonable speech recognition implementation will discard most of the acquired speech without performing feature extraction under this scenario. Therefore, the power consumption can be attributed to the microphone subsystem and speech detection module. From the power trace, we observed that Voice Command calls the speech detection module about every $250ms$. Each call contributes to a peak in the power trace, leading to an average additional power consumption of $126mW$.

Speech acquisition with speech being detected: We next evaluated Voice Command when fed with irrelevant utterances, which are detected as speech but not recognized. The power trace generated was very similar to the one when there was no speech except that the peaks became wider when the input utterances became more continuous. These wider peaks can be attributed to feature extraction performed immediately after speech is detected and recognition decoding after a certain amount of speech is detected. The typical power consumption for processing a continuous irrelevant utterance is about $780mW$. Interestingly, if the utterance is relevant or recognizable, the average

power consumption is actually much lower. For all the traces we obtained with a valid command, the power consumption is usually about $680mW$ in this case. The higher power consumption with irrelevant utterances may be introduced by a larger search space. For valid utterances, the search space can be significantly pruned because some very promising search paths can be identified early.

Speech synthesis: We recorded the power trace for iPAQ when it synthesized the speech output for speech recognition. The additional power consumption for the speaker subsystem at maximum volume is $181mW$, which is significantly smaller than that shown in Table 5.2. This is due to the fact that the sound clip used for generating the table is a continuous flow of music while the synthesized speech output only uses the speaker subsystem intermittently, leading to a much lower duty cycle. The non-speaker subsystem power for speech synthesis is about $75mW$. Compared to the $383mW$ additional power required for performing DCT (see Figure 5.1), such a speech synthesis is not computationally demanding on iPAQ at all.

It is worth noting that for many voice commands, the display need not be on. This means that 82 to $526mW$ $(82 + 444)$ power reduction is possible (see Figure 5.1). As we will see in Section 5.5, speech recognition-based interfaces are more energy-efficient in many scenarios only if the display is turned off when compared to several other interfacing technologies.

5.4 Manual input techniques

We next characterize the additional energy consumption for various manual input techniques for text entry. For letter-based input, such as letter recognition and virtual keyboard, we examine the additional energy consumption for inputting a letter; for word-based input, such as *Transcriber*, we examine the additional energy consumption for inputting words of different lengths. Table 5.3 presents the additional energy consumption for inputting a letter. Figure 5.2 presents the additional energy consumption for inputting words of different lengths using *Transcriber* on iPAQ. The energy consumption per letter increases slightly as the word becomes longer due to a larger recognition effort.

The above text-entry methods consume energy through touch-screen usage and related CPU activities. However, the energy thus consumed is insignificant compared to that consumed by the

Table 5.3: Additional energy consumption for inputting a letter

Input method	Additional energy (mJ)	
	iPAQ	Zaurus
Hardware keyboard	∼30	∼50
Virtual keyboard	∼10	∼80
Letter recognition	∼30	∼330

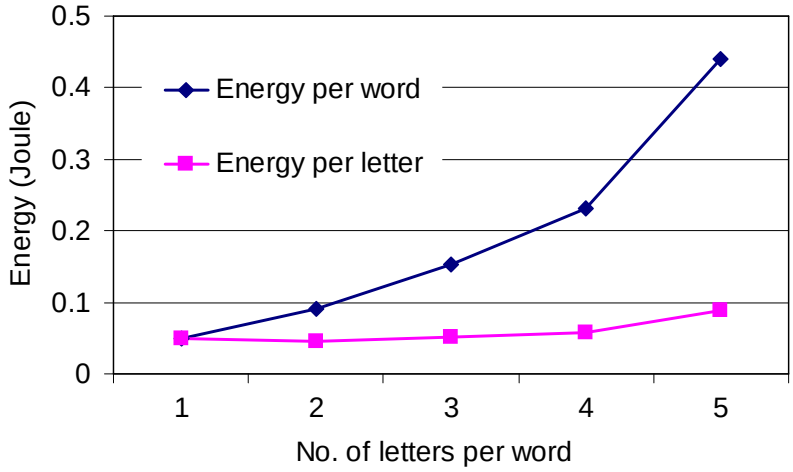


Figure 5.2: Additional energy per word/letter for *Transcriber*

LCD, which needs to be on during text entry. Therefore, the energy cost per letter is not the only indicator of the energy efficiency of a text entry method. What matters more is the entry speed, as we will see in Section 5.5.

5.5 A comparative study

Based on the discussion of interaction speeds and energy characterization presented in Chapter 3, and Sections 5.2 and 5.3, we next compare the energy efficiency of different user interfaces. As speech-based interfaces are gaining ground, we use such an interface as the baseline.

5.5.1 Output

We first examine the energy efficiency for presenting language-based information through speech or text.

When the information to be presented is long enough, the reading/speaking rate determines the duration of presentation. Let R_{spk} denote a comfortable speaking rate and R_{rd} a comfortable reading rate in wpm. Let P_{txt} denote the system power consumption for presenting text. For simplicity, we assume that P_{txt} is roughly constant for presenting any text. We ignore the energy consumed to render the GUI for the text. The computer is basically idle after the text is presented on the display. On the contrary, the computer has to be active when the text is spoken back to the user. Let P_{spk} denote the corresponding system power consumption.

The ratio of energy consumption for text and speech outputs is therefore

$$r_{output} = \frac{R_{spk}}{R_{rd}} \cdot \frac{P_{txt}}{P_{spk}}$$

The following techniques can impact r_{output}: P_{spk} can be changed drastically by turning the display on or off or by using an earphone instead of the loudspeaker; P_{txt} can be reduced by employing aggressive power management [16, 233]. Figure 5.3 gives different values of r_{output} for iPAQ and Zaurus under some possible scenarios based on data presented in Sections 5.2 and 5.3. We assume $R_{rd} = 250wpm$ and $R_{spk} = 150wpm$. "Light" indicates that the back light or front light is on and "PM" or "NPM" refers to whether aggressive power management [16, 233] is employed or not. The X-axis denotes whether the display, together with lighting for "Light," is on or off and whether the built-in loudspeaker or earphone is used for speech output. For Zaurus, the direct playback power consumption is used as P_{spk}. Note that the speech output is more energy-efficient if and only if the ratio is greater than 1.

For iPAQ, when the back light is on for night-time text reading, a synthesized speech output through an earphone with the display off would be more energy-efficient than a text output. For Zaurus, a speech output consistently consumes more energy for day-time usage when the front light is not needed. It is more energy-efficient only when the display does not need to be on. Its advantage primarily comes from the fact that the speech output does not mandate that the power-hungry display

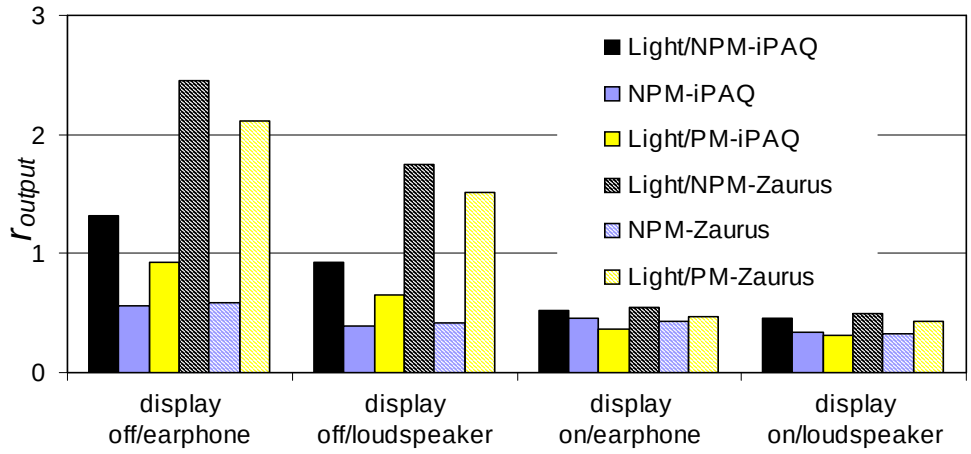


Figure 5.3: Ratio of energy consumptions for text output over speech output under different scenarios

be on. On the other hand, it consumes two to three times more energy if the loudspeaker is used and the display is left on. The key to improving energy efficiency for a speech output is therefore to turn off the display and adopt a low-power audio delivery method other than a loudspeaker.

When the information is very short, such as short messages and notifications, the presentation duration is not primarily determined by the reading/speaking rate but other time overheads for eye/hand movements and distraction. Therefore, speech and audio delivery can be very energy-efficient [62, 179] since it is not visually intrusive and persistent. Such short messages and notifications, if delivered as GUI presentations, could interrupt user's ongoing work and require user action to respond, $e.g.$, to close the popup message box, leading to a larger energy overhead.

5.5.2 Input

Next, we compare the energy efficiency of different input methods. There are two types of input, namely, text and control.

Text entry: In Section 5.4, we derived the additional energy consumption for inputting a letter under different text-entry methods. As pointed out in Chapter 3, the corresponding input speeds vary a lot. Let R_{entry} denote the typical input speed in wpm. Let e denote the additional energy

consumed for inputting one letter using the method characterized in Section 5.4 and P_{idle} denote the system idle-time power consumption. We assume an average word requires six letter inputs [26], including a space. On the other hand, let R_{spk} denote a comfortable speaking rate for recognition-based input and P_{recog} the system power consumption during speech recognition. If we ignore the energy consumed during the delay between the end of speech and the end of speech recognition, the ratio of the energy consumptions for manual text entry and speech-based text entry is given by

$$r_{input} = \frac{\frac{P_{idle}}{R_{entry}} \cdot 60 + e \cdot 6}{\frac{P_{recog}}{R_{spk}} \cdot 60} = \frac{R_{spk} \cdot P_{idle}}{R_{entry} \cdot P_{recog}} + \frac{e \cdot R_{spk}}{10 \cdot P_{recog}}$$

Obviously, the energy efficiency of an input method is primarily determined by its input speed.

Although Voice Command is not intended for text entry, we assume speech recognition-based text entry would have similar power characteristics and therefore use the power consumed by the Voice Command recognition process as P_{recog}. Figure 5.4 plots the r_{input} for the hardware mini-keyboard (HW MKB), virtual keyboard (VKB), and letter recognition (Letter Recog.) using data from Chapter 3, and Sections 5.2 and 5.3. For each method, four cases are shown. "ideal" refers to typical input speed without considering error correction; "No LCD" refers to comparisons to speech recognition with the display off; "No LCD/Light" refers to night-time usage with the back light on as compared to speech recognition with the display off. Except for "ideal," input speeds are expressed in $cwpm$. The break-even line with r_{input} equal to 1 is also shown. For any point above this line, the speech recognition-based input is more energy-efficient. From Figure 5.4, the potential energy advantages for speech recognition-based text input are obvious since speech is potentially much faster than any other text input method. However, a recognition-based input method usually incurs much higher input errors, leading to a much lower speed in $cwpm$. For example, the speed of handwriting recognition is about half the speed of handwriting. Studies in [35] have shown that speech recognition speeds of $17cwpm$ are already available and may reach 45 to $50cwpm$ in the near future. If the power consumption for correcting errors in speech recognition is about the same as the power consumption during recognition, Figure 5.4 shows that speech recognition is already more energy-efficient than letter recognition and also the virtual keyboard for night-time usage if the speech recognition-based interface does not require the display to be on. Moreover, when a speed of 45 to $50cwpm$ is achieved by the speech recognition-based interface, it will be more energy-efficient

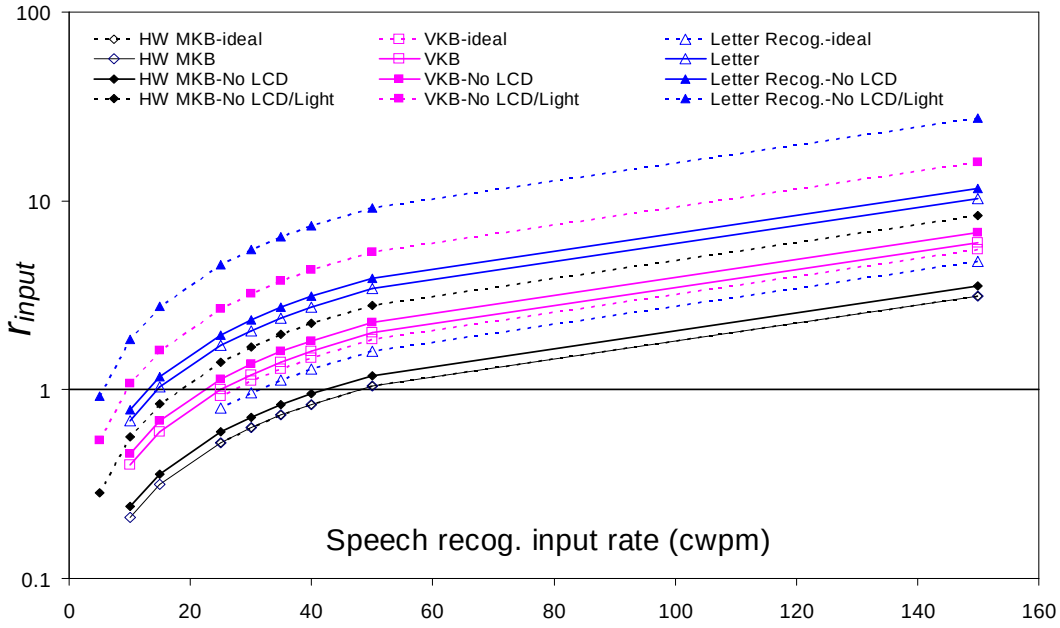


Figure 5.4: Ratio of energy consumption for different text-entry methods over speech-based text entry

than most text-entry methods, even the hardware mini-keyboard. Comprehensive treatments of error correction and speech recognition throughput are provided in [35, 112, 199].

Command and control: Error correction drastically decreases the input speed for speech recognition-based text entry, leading to a much lower energy efficiency. For command/control applications such as Voice Command, however, errors can be corrected much faster, $e.g.$, by reissuing the command. Moreover, for such applications, the recognition accuracy is usually much higher. This leads to a higher throughput and thus a higher energy efficiency. For a command/control task, let us assume it may take M stylus taps or it may take a W-word voice command. Let N denote the speaking rate in $cwpm$. Based on the traces collected for system usage [233], we assume each stylus tap is accompanied by a $750ms$ user delay, which is mostly an underestimation for typical menu selections on iPAQ. Moreover, we assume that the energy consumed by the GUI response can be ignored compared to that consumed during the user delay. Therefore, r_{cc}, which represents the ratio of the energy consumptions by GUI-based and speech-based command/control, is given by:

$$r_{cc} = \frac{P_{idle}}{P_{recog}} \cdot \frac{M \cdot N \cdot 0.75}{60 \cdot W}$$

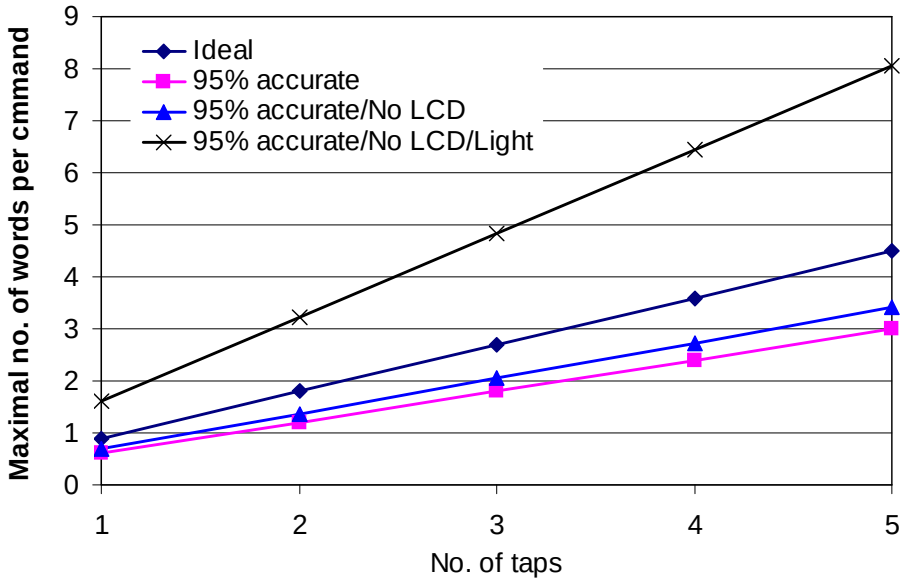


Figure 5.5: The maximal number of words per command for better energy efficiency

Obviously, the shorter a voice command, the more energy-efficient it is. Figure 5.5 shows the maximal number of words per command required so that speech-based command/control is more energy-efficient than GUI operations with different numbers of taps under various scenarios. 100% accurate speech recognition with $N = 150$ is used to draw the "ideal" line. Note that 150 wpm is regarded as the conversational English speaking rate. In other cases, 95% speech recognition accuracy is used with $N = 100$, assuming 10 times more energy/time required to correct an error compared to speech recognition. Such an assumption is pessimistic since most errors can be corrected by simply reissuing the command. "No LCD" and "No LCD/Light" have the same meaning as in Figure 5.4.

Figure 5.5 shows that *a one-word voice command is more energy-efficient than GUI operations with two or more taps*. If the display can be turned off for speech-based command/control, its advantage is higher.

Taking notes: Speech and handwriting recognition-based text entries are mostly hindered by their low accuracy and high cost for correcting errors. However, if text transcription is not needed in real-time, $e.g.$, when using an audio recording or handwriting to take a note, it is most energy-efficient to use speech since speaking is much faster than any other input method. However, if the

note has to be retrieved in the format that it was recorded before recharging, there is a tradeoff between the energy consumptions for taking a note and for retrieving it, especially when it has to be retrieved multiple times.

5.6 Observations

Based on the energy characterization presented in this chapter and Chapter 4, we can make the following observations for improving energy efficiency.

Speed matters: The faster a task is accomplished and the higher the user productivity, the more energy-efficient the system usually is. From this perspective, *interface designers share a significant responsibility for designing an energy-efficient system*. In most cases, improving user productivity may incur average power consumption increase. As long as the productivity improvement percentage is larger than average power increase percentage, the energy efficiency is improved.

In terms of the specific interfacing methods, speech-based input stands out since speech is inherently much faster than other input methods. For recognition-based input, such as handwriting and speech recognition, accuracy is important due to the high cost of correcting errors. Thus, accuracy is also important for energy efficiency.

Display matters: The energy efficiency for a display-based interface suffers a lot since its average power consumption includes that of the display, which is large. Touchscreen/stylus-based interaction basically integrates the input hardware with the output hardware, leading to a high power consumption even for making an input, especially for night-time usage. When the power-hungry display has to be on with a slow input rate, $e.g.$, for all the manual text-entry methods, energy efficiency is drastically reduced. Speech-based interfaces again may enjoy an energy efficiency advantage since their display usage can be carefully avoided.

The power consumption landscape, however, is likely to change in a few years due to progress in new display technologies. OLEDs [58] promise high-quality low-power flexible displays for mobile computers. More importantly, bistable display technologies [40, 114, 229] will reduce the static power consumption to nearly zero. This will significantly reduce the energy that a system spends in waiting for user inputs.

Audio matters: Surprisingly, our energy characterization results showed that the speaker subsystem is also power hungry, drawing as much as $367mW$ and $422mW$ of power for iPAQ and Zaurus, respectively. This power consumption can be drastically reduced by using earphones instead of loudspeakers. However, the wires connecting the earphones may impact other usage issues. Since iPAQ Bluetooth consumes more than $470mW$ additional power when actively transmitting data (see "BT Trans." in Figure 5.1), a Bluetooth headset is unlikely to reduce the audio delivery power consumption. Therefore, *for better exploiting the speed of speech-based interfaces, low-power wireless voice stream delivery between the user and computer is critical.*

Amdahl's Law for energy: Amdahl's Law has been widely used for guiding performance improvement [85]. It indicates that we need to "make the common case fast" [85]. The same law can be easily extended to energy reduction. Suppose the energy consumption for a user task comes from many sources. A particular source contributes to a $Fraction$ of the total. If we can reduce the energy consumption from that source by a percentage, termed $Reduction$, we can reduce the energy consumption for that task by

$$Reduction_{overall} = Fraction \times Reduction.$$

Such a law indicates we should focus on the larger sources. There are two implications. First, we should focus on the system components that consume more energy. For example, we should focus on the display instead of the processor for most GUI-based tasks on mobile systems. Second, we should focus on the time interval when most energy is spent. For example, we should focus on system idle periods instead of busy periods for most interactive tasks, because the system spends most of its energy in idle periods [233]. This also reaffirms our conclusion in Section 5.2 on improving the energy efficiency of the CF digital camera. These two implications are extremely important for interactive systems that engage users through user interfaces since conventional low-power research has focused on how to make computing more power-efficient.

5.7 Chapter summary

This chapter presented energy characterization of state-of-the-art user interfaces on two commercial mobile systems. Based on energy characterization and user productivity information, it offered a comparative study of these interfaces. Specifically, it found that speech-based interfaces have a large potential to outperform other input methods because a human user can speak much faster than write or type. On the other hand, a speech-based output suffers from high power consumption required for audio delivery without enjoying a significant speed advantage over text-based output. Along with Chapter 4, this chapter demonstrated the significant energy efficiency impact of user interfaces. They not only showed that state-of-the-art user interfaces consume far more energy/power than the theoretical minimal as presented in Chapter 3, but also highlighted the slow-user problem as first discussed in Chapter 1. That is, low user productivity (speed) on mobile systems has become a great challenge to their energy efficiency; energy consumption in idle periods is the bottleneck. Tackling the slow-user problem will be the focus of the following three chapters.

Chapter 6

Pervasive Interfacing: A Personal-Area Network of Wireless Interfacing Devices

Chapters 3 to 5 have already highlighted the slow-user problem that severely limits the energy efficiency of mobile systems. On the other hand, as argued in Chapter 1, mobile systems are on their evolutionary track to serve people for pervasive computing, connectivity, and entertainment. Unfortunately, not only has the slow-user problem become a great bottleneck to this goal, but also to solve it means to provide a user with natural access to his or her mobile system anywhere and anytime in an energy-efficient fashion. We call the solution *pervasive interfacing*, which is the focus of this chapter. In this chapter, we present a Bluetooth-based PAN of wireless interfacing devices for pervasive interfacing. The chapter is organized as follows. We set out the design principles in Section 6.1, and then describe the Bluetooth-based PAN and its power optimization in Sections 6.2 and 6.3, respectively. Since we leave the cache-watch for Chapter 7, we provide details of the other two devices, violin-pad and smart speech portal, in Sections 6.4 and 6.5, respectively. After that, we describe how information-capturing devices, such as a Bluetooth global positioning system (GPS) receiver, can be included in the PAN in Section 6.6. We address works related to our system in Section 6.7 and conclude in Section 6.8. Although user studies are critical for such a system that is intended for interfacing, we focus on the *system and hardware design issues* in this dissertation.

6.1 Design principles

In view of the limitations of the current interfacing paradigm, we follow a number of principles for developing new interfacing devices:

- Separating interfacing from computing;
- Separating information capturing from storage;
- Employing more interaction channels such as speech;
- Simplicity: low-power and inexpensive designs.

Most current interfacing paradigms require a user to hold a mobile system in hand to operate, leading to a great waste of our most powerful tools, our fingers, as well as the most precious computing resource, attention, according to [60, 193]. Henceforth, we treat the first two principles as critical. We follow them by employing Bluetooth to connect a mobile system and its interfacing devices into a wireless PAN. We designed a wireless keypad, called *violin-pad*, which is small enough to be attached to a keychain. Since it is physically detached from the mobile system, the user can hold it in one hand and type in a fashion similar to playing violin. We also designed a wrist-watch, called *cache-watch*, which talks to the mobile system from time to time to display critical information on the wrist. Although the use of Bluetooth increases interface power consumption of a mobile system, the latter's energy efficiency will still be improved if user productivity is improved accordingly, as discussed in Section 2.2.2.

As Chapters 3 and 5 demonstrate, human users have very different speeds for different interaction methods. The most popular methods, i.e., keyboard, stylus, and handwriting, suffer great speed limitations due to the human factors. It is, therefore, critical to employ more interaction channels that human beings excel at. Speech, as shown in Chapter 5, is a promising candidate. As mentioned in that chapter, however, speech-based interfaces face a great challenge in delivering a noise-resilient and high-quality voice stream to and from the mobile system in a user-friendly fashion. We design a *smart speech portal* based on bone-conduction sensing to respond to this challenge.

Figure 6.1: System overview of the PAN

As to the last principle, we believe that new interfacing devices have to be inexpensive to generate a market and train users. They have to be low-power to obviate frequent rechargings. We design the wireless interfacing devices as inexpensive add-on devices to the mobile system. For each add-on device, we partition the interfacing task so that only the minimal set of functionalities remain on it. The data processing, control and application-layer protocols are implemented with only one or two Microchip mid-range micro-controllers.

Figure 6.1 shows a system view of our PAN of interfacing devices for a mobile system. The mobile system is the PAN center, called the *digital hub*. The PAN manager and device managers are middleware installed on the mobile system. The PAN manager coordinates communications between the mobile system and its add-on devices, while a device manager interprets raw data received from the corresponding add-on device, controls it, and functions like its device driver.

6.2 Bluetooth-based PAN

In this section, we provide details of the PAN that integrates a mobile system and its wireless interfacing devices.

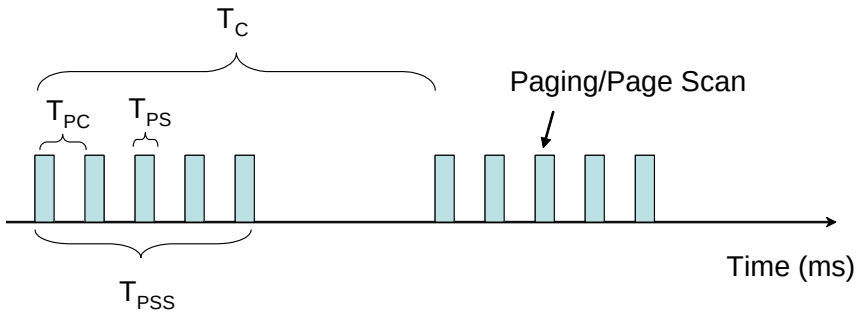


Figure 6.2: Paging/Page-Scan session

6.2.1 Bluetooth

The Bluetooth standard [15] provides a number of profiles for different applications. For example, most Bluetooth headsets implement the Headset Profile, which provides a two-way 64kbps (kilo-bit per second) voice stream. Using the profiles simplifies application development at the expense of a loss in flexibility and increase in hardware complexity. Since we wish to simplify the add-on devices as much as possible, we choose the simplest profile, the Serial Port Profile (SPP), for all devices. The add-on devices only send out and receive raw data through SPP, while the device managers installed on the mobile system interpret the data accordingly.

Establishing a connection: To establish a connection between two Bluetooth devices, one of them has to initiate the connection by *Paging*. It is called the *initiator*. The other device must do *Page Scan* to accept the initiation and establish the connection. It is called the *responder*.

Both Paging and Page Scan are carried out in sessions. In a Paging/Page-Scan session, a Bluetooth device does Paging/Page-Scan for T_{PS} seconds every T_{PC} seconds. T_{PS}, T_{PC}, and the length of a session, T_{PSS}, can be changed by software. Usually, T_{PS} and T_{PC} are multiples of a 625μs slot and are on the order of milliseconds and T_{PSS} is on the order of seconds. Figure 6.2 shows that a Bluetooth device enters a Paging/Page-Scan session every T_C seconds.

In our PAN of interfacing devices, both the initiator and responder specify the Bluetooth address that they wish to connect to. The mobile system can be either an initiator or responder. The smart speech portal and the violin-pad initiate the connection with the mobile system upon user's instruc-

tion. Such devices are called *active* devices since they require user engagement to function. The cache-watch, GPS receiver and other information-capturing devices are responders. They are called *passive* devices since they function under the control of the mobile system without user engagement.

iPAQ Bluetooth: The HP iPAQ Pocket PC 4350 described in Section 5.1 is the mobile system in our prototype system. It is equipped with Windows CE Pocket PC 2003 edition and a Bluetooth protocol stack from WIDCOMM (acquired by BROADCOM). The Bluetooth protocol stack emulates two virtual serial ports, each for incoming and outgoing connections. Since a port cannot communicate with more than one Bluetooth device simultaneously, our mobile system is able to communicate with only two devices at the same time, which is a limitation of this implementation. Since we have more than two devices, a time-division access scheme has been implemented. Note that "incoming" and "outgoing" only refer to which party initiates the connection. Both ports are full duplex.

Bluetooth-RS232 adapter: The adapters, called Promi-ESD^{TM}, used in the interfacing devices are manufactured by Initium [99] and belong to two classes. Class I adapters have larger transmission power than class II ones as specified by the Bluetooth Standard v1.1. They include a low-power implementation of the Bluetooth SPP. The low-power Park mode is supported only for a connection between two Promi-ESD^{TM} modules, which is not true in our project. Other Bluetooth low-power modes are not supported at all. As a result, we have to rely on the supported APIs to configure the parameters for Page Scan/Paging and connect/disconnect for power management. A Promi-ESD^{TM} module can be configured so that it connects to or can be connected from a certain Bluetooth address.

Power characteristics: Figure 6.3 shows the power characteristics for the Promi-ESD^{TM} class I and iPAQ 4350 Bluetooth modules. In the STANDBY mode, the Bluetooth module is powered-on but not active. There are no data for the iPAQ Bluetooth STANDBY mode since we were not able to put the iPAQ Bluetooth module into the STANDBY mode. In the PENDING mode, the Bluetooth module is in a Paging/Page-Scan session. Note that we use the default values for T_{PC} (1024 slots) and T_{PS} (128 slots), which have an impact on the power consumption in the PENDING mode. In the CONNECTED mode, the Bluetooth module is connected with another Bluetooth device, ready

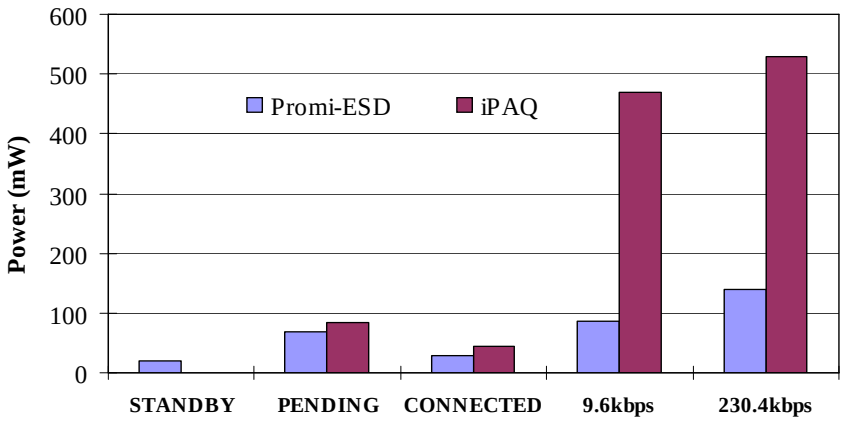


Figure 6.3: Bluetooth power consumption

for data transmission. Figure 6.3 also shows the power consumed when the Bluetooth module is connected and transferring data at $9.6kbps$ and $230.4kbps$.

The data for Promi-ESD^{TM} class I module are taken from the manufacturer's manual except that the power consumption for $230.4kbps$ is based on our measurement. Those for class II modules are similar. The power consumption for the iPAQ 4350 Bluetooth module was also measured. All the power measurements were based on measurements of the voltage drop across a $100m\Omega$ sense resistor embedded in the power supply. The voltage drop was measured with an Agilent 34401A digital multi-meter. The power consumption of the iPAQ Bluetooth module shown in Figure 6.3 is actually the extra power consumption, which is the difference between the power consumption of an idle iPAQ with Bluetooth off and that of the same iPAQ with Bluetooth in different modes. The results show that the Promi-ESD^{TM} is much more power-efficient than the iPAQ since the former has a much more efficient hardware implementation.

6.2.2 PAN manager

The PAN manager on the mobile system is critical to the functions of the PAN. It was developed using Microsoft embedded Visual C++ and the BTAccess library [19]. It consists of three major components and interacts with the device managers and Bluetooth protocol stack, as illustrated in

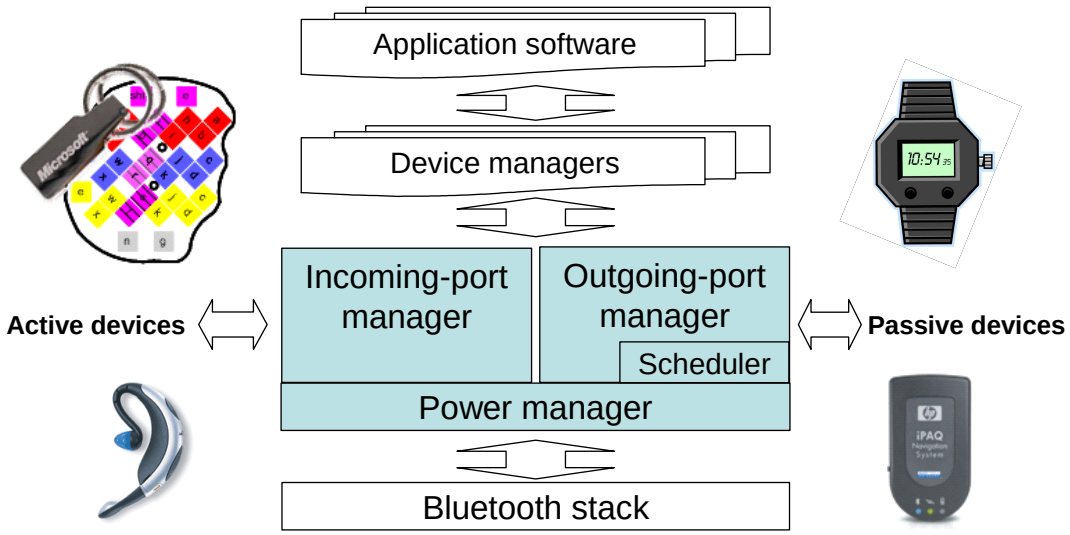


Figure 6.4: Software installed on iPAQ; the PAN manager is shaded

Figure 6.4. When a new device joins the PAN for the first time, it has to register with the PAN manager through a Bluetooth Discovery process. After that, both the add-on device and the PAN manager remember each other's unique Bluetooth address. The PAN manager also gets information about the add-on device's type and device manager. After being registered, the add-on device can join the PAN by connecting with the mobile system directly without the standard Bluetooth Discovery process.

Incoming-port manager: The incoming-port manager controls the connections of the mobile system with active devices, such as the smart speech portal and violin-pad. It forces the mobile system to enter a Page-Scan session periodically since requests for connection from active devices should be served timely. In our prototype, we use the default values of T_{PC} and T_{PS}, and choose T_C and T_{PSS} to be 4 and 2 seconds, respectively. There is a tradeoff between energy consumption and connection delay involved. We address this issue in Section 6.3. The user can also manually stop or start the manager to save energy of the mobile system. Once a connection is established, the corresponding device manager is called.

Outgoing-port manager: Passive devices, such as the cache-watch and information-capturing devices, connect to the outgoing-port as responders. Unlike active devices, such as the violin-pad, the communication delay between passive devices and the mobile system is usually tolerable. There-

fore, the outgoing-port manager schedules the communication to share the port among all passive devices. The device managers for those passive devices run even when there is no connection. They interact with applications, such as Outlook, and send requests for a connection to the scheduler. The port manager buffers these requests, surveys the PAN members periodically to check which passive devices are available, and schedules its outgoing connection accordingly. When a connection to a passive device is scheduled, the outgoing-port manager forces the mobile system to enter and stay in a Paging session for a certain period of time until the connection is established.

Power manager: The power manager is responsible for system and Bluetooth power management of the mobile system. Ideally, the power manager should be able to place the Bluetooth module into different power-saving modes based on information from the incoming-port and outgoing-port managers. Unfortunately, since we do not have direct access to the Bluetooth protocol stack on iPAQ, the power manager in the prototype can only turn the Bluetooth on and off for power savings.

6.3 System power optimization

Battery lifetime is critical to all devices involved in the PAN. For the mobile system, Bluetooth is one of the largest power consumers. It inflicts the primary power overhead on the mobile system for using a wireless PAN. For the interfacing device prototypes, the Bluetooth modules dominate the power consumption, usually taking more than 90% of the total. Therefore, we focus on Bluetooth for power optimization.

Scheduled communication for passive devices: For a passive device, such as the cache-watch and GPS receiver, Bluetooth consumes most of its energy in the Page-Scan sessions since data communication is usually very brief. Fortunately, as mentioned before, the mobile system knows when it needs to talk to a passive device with information from the outgoing-port scheduler. Therefore, when the mobile system is connected to a passive device, the outgoing-port manager predicts when the mobile system will seek communication with the passive device again based on prior history or a prefixed schedule. The mobile system then notifies the passive device just before the end of the current communication. The passive device will shut its Bluetooth module down or put it into the STANDBY mode, depending on when the next communication is scheduled. Later on, both the mo-

bile system and passive device enter the PENDING mode just before the scheduled communication so that both will spend minimal time in the PENDING mode. Only when they lose synchronization do they enter the PENDING mode periodically to re-synchronize.

Tradeoffs for active devices: Active devices, such as the violin-pad and smart speech portal, typically seek connection with the mobile system when the user explicitly makes such a request. The connection latency is important to the user. Since user behavior is very hard to predict in this case, to guarantee minimal latency, the mobile system Bluetooth would need to stay in the PENDING mode all the time, which will drain the battery very quickly. Therefore, we trade connection latency for energy saving on the mobile system. Instead of staying in the PENDING mode, the mobile system enters it and stays in it for 2 seconds every 4 seconds. With an extra latency of 2 seconds, we reduce the energy consumption by half. Moreover, users can manually start or stop the mobile system Bluetooth Page-Scan session if more energy savings are desired.

Real-time data compression: Smaller the amount of data transferred, smaller is the energy consumption. However, data compression can only benefit data-intensive devices. The only data-intensive device in the PAN is the smart speech portal. We design a simple real-time differential speech coding scheme to reduce its data rate and energy consumption.

Limitations: Despite the above optimizations, we still suffer from limitations imposed by Bluetooth itself and its implementations on iPAQ and Promi-ESD^{TM}. Establishing a connection is a lengthy and power-hungry process for Bluetooth. We have observed that it consumes most of the energy for passive devices. When we started our project, only two wireless PAN technologies were commercially available: Bluetooth and ZigBee. We chose Bluetooth over ZigBee for two reasons. First, a variety of Bluetooth modules are widely available and high-end PDAs/smart-phones have already been equipped with Bluetooth. Second, ZigBee does not provide a data rate high enough for media applications we envisioned for the future. We are now evaluating the possibility of a heterogeneous wireless PAN with ZigBee for low data-rate applications and Bluetooth for high data-rate ones. The Bluetooth implementation on the iPAQ also severely limits its energy efficiency. The Bluetooth protocol stack is implemented in software. That is, the whole system except the display has to be on if Bluetooth is needed. A much better approach would be to implement the Bluetooth

(a) Prototype (b) Usage

Figure 6.5: The violin-pad

protocol stack on a separate system so that Bluetooth can operate without the whole system being active and the system can be woken up based on Bluetooth activities. The lack of support for low-power modes in Promi-ESD^{TM} is also a limiting factor. For the current prototypes, we have to shut down the Promi-ESD^{TM} modules to save energy, and pay the price in terms of latency and energy overhead for reconnecting. In summary, the energy efficiency of the whole PAN can be significantly improved based on better wireless modules.

6.4 Violin-pad

As shown in Chapters 3 and 5, slow text entry is an obstacle to high energy efficiency and new services on mobile systems. In this section, we present our solution to this problem, a keypad inspired by the violin, called violin-pad.

Design: Figure 6.5(a) shows the violin-pad prototype. The electrical part consists of two modules: Promi-ESD^{TM} class II and Microchip PIC16LF873. Although it is the electrically simplest one of the three devices, the violin-pad is mechanically the most challenging due to its size. We have to make sure that proper tactile feedback is provided to the user and the buttons are properly designed to achieve a decent input speed with a low error rate. 12 of the 20 mechanical buttons are two-way rocker-buttons, i.e., they can be pressed toward either the top or the bottom end with different directions corresponding to different inputs. The button design used in this prototype is illustrated in Figure 6.6. Such buttons are called *double buttons.*

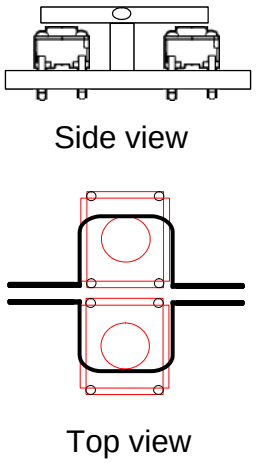


Figure 6.6: Double button design

The alphabet and digit input relies on the 12 double and 4 regular buttons. They are organized into four columns, each akin to a violin string, and four rows, to each of which a finger is devoted. The double buttons in the third column have a prominent texture so that the user can feel which column the fingers are on. For the current version, the keys are organized primarily according to the alphabet. Since the user's hand is in control when the violin-pad is used, there is a button for the user to switch it on/off. Upon being switched on, the violin-pad enters a Paging session to connect to the mobile system. It draws about 33mW power during active usage.

Usage: The violin-pad is intended to be used in a fashion similar to how a violinist positions fingers on the violin neck. That is, the user holds the pad in one hand and uses four fingers for active typing, as illustrated in Figure 6.5(b). However, the user can use it in any other preferred way. We are still conducting user studies of the violin-pad with initial results being very promising. In addition to providing fast text-entry, the violin-pad can be used together with the cache-watch for short-message inputs without the user having to take out the mobile system. These two devices can communicate through the mobile system at the same time because they connect to different iPAQ Bluetooth virtual serial ports.

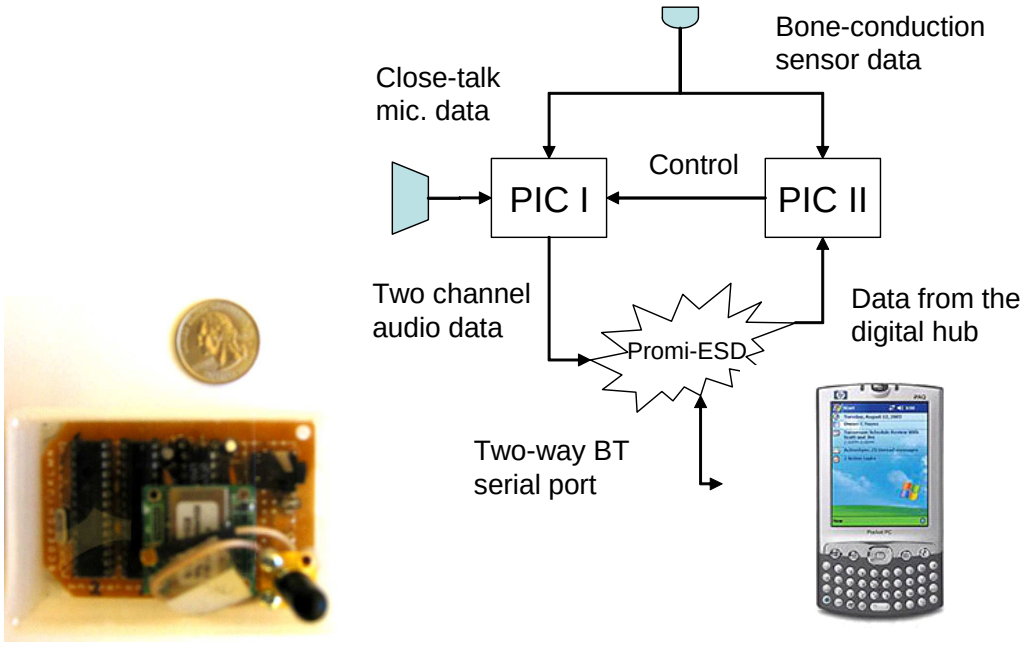
The main concern with respect to the violin-pad and other active devices is the connection latency with the mobile system. As mentioned in Section 6.3, the mobile system enters a Page-

Scan session only periodically for reducing Bluetooth-related energy consumption. After the user requests a connection, there may be a latency of up to two seconds before violin-pad's Paging actually catches mobile system's Page Scan. Then it takes about three more seconds to establish the connection. That is, the user has to expect a latency of three to five seconds. Latency improvement without power overhead is possible only if more Bluetooth low-power modes are supported.

Related devices: Text-entry for a highly mobile device is challenging due to the available real-estate. Starner offers an excellent survey in [197] on this topic. It is no surprise at all that many existing solutions are not satisfactory. Many suffer from very low speed, such as Multi-tap (used on cell phones), Thumbscript [206] and Fastap [46]. They share the same two problems. First, they are physically attached to the host device. The user has to hold the mobile system to type. Second, the user can at most employ two fingers for typing. The violin-pad solves these two problems by using Bluetooth and a violin-neck-like design. The design of Twiddler [137, 209] could look similar to the violin-pad. However, cording, which is key to Twiddler, imposes a significant memory load on its user. On the contrary, the violin-pad is based on single-tap instead of cording. Moreover, Twiddler is wired instead of being wireless, and was not designed for a size that can fit into a keychain.

6.5 Smart speech portal

The most challenging device of the three is the smart speech portal. Indeed, there is a Bluetooth Headset Profile, which was intended for voice communication instead of speech interface. The supported audio quality (8K samples per sec. and 8 bits per sample) is not recognition-friendly or noise-resilient. Moreover, it becomes difficult to implement features facilitating speech recognition if the Headset Profile is used. In this section, we provide details of the smart speech portal. Based on the simplest Serial Port Profile, it offers better audio quality (11K samples per sec. and 10 bits per sample) and uses a bone-conduction sensor to provide a clean voice stream for recognition. With the smart speech portal, the user can just talk to establish a connection between the speech portal and mobile system, send speech data in a real-time fashion to the mobile system, and leverage the computing power on the mobile system for a speech interface. Extreme care has been taken to pack speech and bone-conduction data into the $230kbps$ Bluetooth serial port and implement power



(a) Prototype (b) Data and control flow

Figure 6.7: The smart speech portal

management using only minimal hardware and computing power.

Hardware design: Figure 6.7(a) shows a prototype of the smart speech portal. Note that the close-talk microphone and the bone-conduction sensor are plug-ins not shown in this picture.

The prototype consists of two Microchip PIC16LF873 (PIC I and PIC II), one Promi-ESD^{TM} class I module, and one operational amplifier module (National LM6134). The Promi-ESD^{TM} class I module is used because we found that data loss is significant for class II modules when the data rate is high. PIC I is devoted to packing data and sending them to the mobile system; PIC II is in charge of speech detection using bone-conduction sensing, receiving data from the mobile system and power management. A simplified block diagram is offered in Figure 6.7(b). The prototype draws about 23mW when Bluetooth is disconnected but not off, about 33mW when connected but the user is not talking, and about 125mW when the user is talking.

Software design: PIC II implements a basic speech detection scheme based on zero-cross rate and energy of the bone-conduction signal [177]. This information is conveyed to PIC I through the *speaking* control signal. It also determines how long the user has either been talking or remained

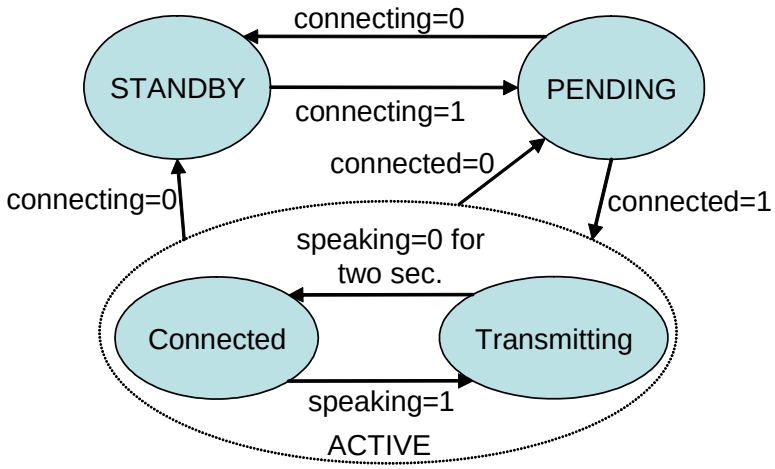


Figure 6.8: State-machine description of PIC I software

silent. When the Bluetooth is disconnected and PIC II finds the user has been talking for a certain period of time (two seconds in the prototype), it raises the *connecting* control signal to PIC I. When Bluetooth is connected and PIC II finds the user has remained silent for a certain period of time (10 seconds in the prototype), it lowers the connecting signal. PIC II also obtains the status information of the Bluetooth module and conveys it to PIC I via the *connected* control signal. PIC II is also in charge of receiving and interpreting data from the mobile system via the Bluetooth module. With speech recognition on the mobile system, a user can use voice command to control the speech portal such as turning it off.

Software on PIC I is best illustrated by Figure 6.8. Each state in the state machine corresponds to a different Bluetooth mode. When PIC I is in STANDBY, the Bluetooth module can be shut down or put into the STANDBY mode. Upon receiving the connecting signal, PIC I enters the PENDING mode and sends commands to the Bluetooth module to start a Paging session. When the connected signal is high, PIC I enters the ACTIVE mode, which has two sub-modes. When the speaking signal is on, it starts analog-to-digital conversion and data coding/transfer immediately (*Transmitting* sub-mode); if the speaking signal is off for two seconds, it stops data transfer to the Bluetooth module (*Connected* sub-mode), which drastically reduces the power consumption. Just before stopping, it also sends a special series of data to the mobile system so that the latter knows that transmission is about to pause. This is useful when a time stamp is needed for each segment of

speech. The data from the bone-conduction sensor can be used to suppress noise [231]. They also provide robust information on the user's oral activity. We have been investigating how to utilize such an information for interfacing. For this purpose, the smart speech portal also sends the bone-conduction data to the mobile system for further processing. Since the bone-conduction signal is low-pass filtered, it is sampled at a rate half of that for the close-talk microphone. PIC I utilizes a simple differential speech coding algorithm to reduce data rate and power consumption.

Device manager and applications: The device manager on the mobile system is responsible for decoding the data into a continuous voice stream, which can either be recorded or supplied to a speech recognizer. It also detects whether data loss has occurred in transmission. With the prototype, a user can connect the smart speech portal to the mobile system by talking for a few seconds. The user can disconnect by either remaining silent for a while or simply uttering the corresponding voice command. While the smart speech portal is connected to the mobile system, the user can access the mobile system's computing power in a hands-free fashion through robust command and control speech recognition.

There have been research activities to record a user's daily activity as the digital diary [64]. A digital camera has already been prototyped for this application [65]. Because personal voice is an integral part of personal life, the smart speech portal can also be employed for digital diary-like applications. Since the smart speech portal only records data when the user is talking, it automatically solves the privacy and data volume problem that many digital diary recording devices may face.

Related devices: Sensors and bone-vibration have long been used for enhancing speech quality and improving speech recognition [78, 231]. None of the known devices are wireless or designed for mobile systems. Bluetooth Headset Profile-based wireless headsets have been commercially available. Again, all these devices are intended for voice communication instead of speech-based interfaces. The Nomadic Radio system [183] is a wireless LAN-based feature-rich audio interface. The authors focus more on the system than the interfacing device. Therefore, for the hardware device, power-hungry wireless microphones and stereo transmitters are used. Moreover, the system is intended to work with PCs instead of a mobile system.

6.6 Information-capturing devices

In last two sections, we provided details on two of the wireless interfacing devices designed to join a mobile system-centered PAN for pervasive interfacing. In this section, we address how information-capturing devices can be added to the mobile system-centered PAN to enable more services.

Information-capturing devices such as sensors and GPS receivers are passive devices in our system. Like the cache-watch, they are connected by the mobile system using scheduled communication as described in Section 6.3. They also abide by instructions from it for power and information management. We have incorporated a Bluetooth GPS receiver into the system. The GPS receiver connects to the iPAQ Bluetooth outgoing port as a passive device and sends data to the iPAQ periodically. Since the GPS receiver was purchased off-the-shelf, we were not able to program it for power-management schemes such as those used in the cache-watch. Instead, the PAN manager in the mobile system schedules the connection with it and the cache-watch. Once connected, the mobile system reads information from it and disconnects immediately. In this way, the mobile system is able to collect the user's position information periodically. If the GPS receiver's firmware could be modified, the scheduled communication could be readily used for energy saving. Other information-capturing devices, such as cameras and sensors for user physiological information, can be incorporated into the PAN in a similar way. Therefore, such a mobile system-centered PAN provides an ideal platform for context-aware applications, and personal services such as digital diary and continuous health monitoring.

The issue with sensors is that a Bluetooth module can be too expensive in terms of power consumption, as discussed in Section 6.3. There are two ways to solve this problem. Some sensors can be embedded on other add-on devices. For example, the sensors for skin moisture, pulse rate, and body temperature can be mounted on the cache-watch to share the Bluetooth module. The other solution is to implement a heterogeneous PAN with both Bluetooth and ZigBee, as mentioned in Section 6.3.

6.7 Related work

As works related to each interfacing device have been addressed in the corresponding section, we address works related to our system: the mobile system-centered PAN of interfacing devices. PAN technologies have been investigated for many years. Using PAN for pervasive/wearable computing is a natural choice since devices serving the user have to communicate with each other. Bluetooth-based desktop sets, including a mouse and a full-size keyboard, have been commercially available for some years. However, such desktop sets were intended to eliminate cords instead of serving pervasive computing.

In 2003, Motorola unveiled its Offspring concept design [150], in which a mobile system, called wireless digital assistant (WDA), connects to a number of interfacing devices through Bluetooth. However, there have been no updates on the design since then. IBM Zurich Lab also has an ongoing project called “Personal Mobile Hub” [94] that consists of a mobile system-centered PAN. These works and ours share the vision for the future of mobile systems. However, our work is focused on how to access the computing power on the digital hub pervasively.

The Spartan BodyNet [49] is most related to our work. The Spartan BodyNet is a Berkeley Dot Mote-based low data-rate wireless PAN with three interfacing devices. The TiltType wrist device is a tilting input device in the form of a watch. The notification ring use two LEDs to attract user attention, and a phicon authorizes a certain level of functionality from the PAN. Since the authors are more interested in describing each device, they offered no detail on how the PAN is managed. On the contrary, we emphasize the importance of PAN design and its management following our guiding principles. Moreover, the devices designed for the Spartan BodyNet offer the user very limited capacity to access the computing power on the mobile system. Indeed, they are not intended to provide pervasive interfacing capacity. On the contrary, our devices are designed explicitly for this goal.

6.8 Chapter summary

This chapter presented the design and prototype of a mobile system-centered PAN of low-power wireless interfacing devices, as our first step toward pervasive interfacing and solving the slow-user problem. The PAN improves the energy efficiency of a mobile system through improving user productivity without increasing power consumption much. It is centrally managed by the mobile system, and interfacing devices can join as add-on devices. Three add-on devices have been designed and prototyped for this purpose in this work. The violin-pad and the cache-watch provide the mobile system the two basic means of interfacing, text-entry and display, at the same time. The smart speech portal provides a hands-free speech portal to support the maturing speech recognition technology. With these devices, the user does not have to physically access or operate the mobile system to use its computing power, unless there is a need for the larger display. The PAN also provides a platform on which new services, such as personal digital diary and continuous personal health monitoring, can be based.

Chapter 7

Interface Cache

The essence of the slow-user problem is the speed gap between an increasingly powerful computer and a constantly slow user. Chapter 6 addressed this speed gap by improving user speed without increasing power consumption much. Unfortunately, user speed is eventually limited by human factors. The approach of Chapter 6 has its own limits. Even with perfect user interfaces, a mobile system will still face the slow-user problem, spending most of its time waiting for a constantly slow user. On the other hand, as Chapters 3 to 5 clearly show, energy requirements of state-of-the-art interfaces are far from the theoretical minimal. In fact, interfacing components, such as the display and speaker subsystems, are among the most power-consuming components. Since interfacing components have to be on when interacting with the slow user, they aggravate the slow-user problem. Realizing that human speed is essentially limited, this chapter takes an opposite yet complementary approach: reducing interfacing power consumption without reducing user productivity much. The chapter is organized as follows. We first offer the motivation in Section 7.1, and then present the design and prototype of the cache-watch in Section 7.2. The cache-watch was first introduced as an add-on device to the PAN in Chapter 6. We will, however, now focus on its role as an interface cache for a mobile system. We provide the evaluation of the cache-watch and address the general design issues of interface cache devices in Sections 7.3 and 7.4, respectively. We discuss related work in Section 7.5 and summarize the chapter in Section 7.6.

7.1 Interface cache

Within a computer, there is another speed mismatch, which is between the processor and memory. This problem is partially alleviated by using a cache [85], a much smaller but faster memory unit, to hold frequently-accessed data. A similar idea can be employed to address the speed mismatch between the user and computer as well, with the cost to be reduced being power consumption and its capability measured as capacity to accommodate interactive tasks from the host computer. This motivated us to design a low-power wireless device, to which the host computer can outsource simple interactive tasks. As an interface solution to the energy efficiency bottleneck due to a slow user, such a device functions like a cache for more expensive interfaces on the host, reducing interfacing energy requirements without sacrificing user productivity much. It saves energy essentially by bringing the interface energy requirements closer to the theoretical minimal without sacrificing user productivity much for the simple outsourced tasks. Table 7.1 summarizes the analogy between the memory cache and interface cache.

Table 7.1: Memory cache vs. interface cache

	Memory cache	Interface cache
Speed mismatch	Processor & memory	Computer & user
Task to outsource	Frequently accessed data	Frequent interactive tasks
Cost to reduce	Memory access latency	Interfacing power

7.2 Bluetooth-based cache-watch

Our cache device takes the form of a wrist-watch that communicates with a mobile system wirelessly. It is the cache-watch introduced in Chapter 6 as an add-on device to the Bluetooth-based PAN. The cache-watch prototype and its host, the iPAQ used in Chapters 5 and 6, are shown in Figure 7.1.

Figure 7.1: The prototype of a wrist-watch as the interface cache for iPAQ

As an interface cache, the watch provides the host computer with a limited display. It provides two different services: active and passive. The cache-watch stays connected with the host and waits for user input for the active service. The user input can be echoed on the watch in real-time. For passive service, the watch communicates with the host from time to time to receive data. The connection can be closed after data transmission but the user can still access the data cached on the watch. The cache-watch provides its services through a simple application-layer protocol. It is up to the host computer to determine what and how to display and how the connection is managed using the protocol. The cache-watch simply follows instructions from the host computer as a slave.

Hardware components: The cache-watch consists of three major components: a Microchip PIC16LF88 microcontroller, a 2×8 monochrome character LCD, and a Bluetooth-RS232 adapter (Promi-ESD class II) from Initium [99]. One MAX604 linear regulator is also used. The system is powered by a 3.6V supply with three $800mAh$ rechargeable AAA batteries. The microcontroller is run at a $10MHz$ clock frequency, drawing a current of less than $0.6mA$. It drives the LCD module directly but controls the Bluetooth adapter through a $9.6Kbps$ serial port. The LCD module draws a current of about $1mA$. There is no lighting for the LCD module in the prototype, a limitation of the current implementation that can be easily alleviated. Therefore, we assume that the LCD module can be used at night time in the following discussion.

The Bluetooth-RS232 adapter implements the simplest Bluetooth application profile, the Serial

Port Profile (SPP). Two devices with Bluetooth SPP can communicate in the same way they would with an RS232 serial port connection. The Bluetooth adapter has a number of operational modes. When it is not connected or seeking a connection, it is in the STANDBY mode, drawing about $6mA$ current, but can be turned off for more energy savings. In the following discussion, we use the STANDBY mode to refer to both situations. When the Bluetooth adapter is seeking a connection through a Page-Scan session, it is in the PENDING mode, drawing about $19mA$ current. In the PENDING mode, it does Page-Scan for $T_{PS}\ ms$ every $T_{PC}\ ms$. T_{PS} must be a multiple of a $625\mu s$ slot. This was described in detail in Section 6.2.1. Both T_{PS} and T_{PC} can be configured through commands from the RS232 serial port, leading to different average power consumption. When the Bluetooth adapter is connected, it is in the ACTIVE mode, drawing about $9mA$ current for no data transmission and about $28mA$ during active data transmission at $9.6Kbps$.

Communication design: Since the data for the passive service are not time-critical, the host buffers data for a certain period of time and a connection to the host is required only from time to time. The communication between the cache-watch and its host computer is not data-intensive and communication occurs only sporadically. Therefore, energy consumption due to data communication is very small compared to that required to establish a connection. However, based on our energy characterization data presented in Section 7.3, it is more energy-efficient to disconnect and then reestablish a connection in a cooperative fashion when the connection is required more than 30 seconds later. By "cooperative," we mean that both the cache-watch and the host enter the PENDING mode at the same time.

When the host is connected to the cache-watch, it schedules the next communication with the cache-watch according to prior-history-based prediction. It then determines whether the current connection should be maintained or closed based on when the connection will be required the next time. If the connection needs to be closed, the host notifies the cache-watch when it will seek a connection the next time. After receiving such a notification, the cache-watch shuts down its Bluetooth adapter and forces it back into the PENDING mode when the specified time has elapsed. This ensures that a connection is established in a cooperative fashion, and keeps the cache-watch and the host synchronized with a relatively low energy overhead, as we will see in Section 7.3. If the

2 bytes	1 byte	Up to 176 bytes	2 bytes
Magic series I	Command type	Command data	Magic series II
	Instructional	4 bytes	
	Informative	Up to 176 bytes	
	Management	4 bytes	
	Reverse	16 bytes	

Figure 7.2: Command format for the communication protocol

cache-watch loses synchronization with the host, they enter the PENDING mode to re-synchronize.

It is the host computer that is charged with communication scheduling. The cache-watch simply follows commands. Such a task partition again simplifies the design of the cache-watch while offering most flexibility for application development.

Communication protocol: The communication protocol, called the synchronization protocol, for the host and watch has a critical impact on the complexity and functionality of the cache-watch, since the cache-watch basically interprets commands from the host based on the protocol. The protocol is interpreted by PIC. The basic communication unit is a command with the format shown in Figure 7.2. Each command is demarcated by two sets of two bytes each, called the magic series. Command data are formatted and interpreted according to the command type. Commands of the first three types are sent from the host to the cache-watch. The *instructional* commands are used to instruct the cache-watch to disconnect or switch between passive and active services. *Informative* commands are used to update the internal memory of the cache-watch for display. The command data not only specify the text to be displayed but also how it should be displayed. The *management* command notifies the cache-watch about the host schedule for the next connection so that the cache-watch can shut down the Bluetooth adapter beforehand. Unlike the first three types of commands, a *reverse* command is sent from the cache-watch to the host. Every time they are connected, the cache-watch first sends a reverse command to the host to notify the latter about what the user has done with text messages, such as deleting or confirming a text message.

Software design: The software on PIC is developed using PicBasic Pro. Timing is implemented using TIMER1 in PIC, which is configured to generate an interrupt every $52.4288\mu s$. In the interrupt handler, the time and display are updated. Counting 19 interrupts as one second leads to

an inaccuracy of about +4 seconds every 1000 seconds. This inaccuracy is taken into consideration when the cache-watch times itself for the next scheduled connection with the host. In the main loop, PIC reads its hardware UART and interprets the data according to the synchronization protocol. For the passive service, the connection is closed immediately after data exchange and the cache-watch displays the cached messages. For the active service, the connection is maintained until the host or the user instructs it to be closed by an instructional command or a button click. Following instructions from the host or user, PIC can send AT commands to change modes for the Bluetooth adapter.

Software on the iPAQ, called *watch manager*, was developed using Embedded Visual C++ and built upon the BTAccess library [19]. The watch manager functions like a device driver. On the one hand, it implements the synchronization protocol. On the other hand, it collects information from different application software, such as Outlook, according to the user configuration. The information is buffered in the watch manager and then sent to the cache-watch when it connects to the host. Each time it is connected to the cache-watch, the manager schedules the next connection and notifies the cache-watch through a management command.

Interface design: For this prototype, we used very simple interfaces to support the targeted services: a 2×8 monochrome LCD and three tact buttons, Buttons 1, 2, 3. The user can change the cache-watch service mode by clicking Button 3. When the cache-watch is in the passive service mode, a Button 3 click puts the Bluetooth adapter into the PENDING mode unless the connection with the host is established. When the cache-watch is in the active service mode, a Button 3 click simply closes the connection and brings the Bluetooth adapter back to the STANDBY mode to wait for the next scheduled connection.

In the active service mode, a Button 1 click clears the display and sends a negative confirmation back to the host, while a Button 2 click simply sends a positive confirmation. In the passive service mode, the user can use Buttons 1 and 2 to browse the text messages cached in the cache-watch. The interface is better represented as a finite-state machine, as shown in Figure 7.3. In the $auto$ state, the cache-watch displays valid message entries in its message cache by rolling the text messages through the first line of the LCD. Each message is rolled according to its meta-data, which specify

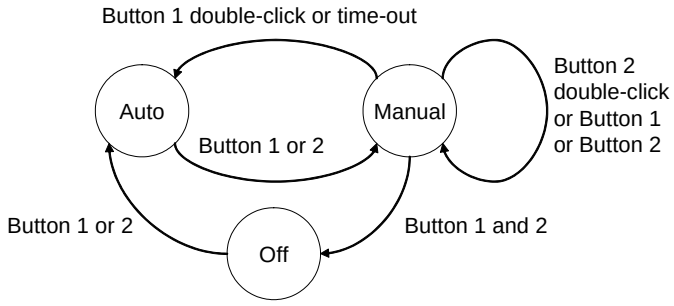


Figure 7.3: State-machine description of the cache-watch interface

its priority in terms of how many times it has to be repeated with each display cycle. In the $manual$ state, the user can use Buttons 1 and 2 to browse valid entries. Clicking Button 1 induces a skip to the next valid entry whereas clicking Button 2 rolls the current message on the LCD by one letter. Double-clicking Button 2 marks the entry currently on the LCD as invalid and confirmed, which is conveyed to the host via a reverse command next time the cache-watch gets connected to the host.

7.3 Evaluation

We evaluated the prototype watch device as the wireless cache for iPAQ in order to see whether or by how much it would improve the battery lifetime of iPAQ. Instead of using user studies and subjective metrics, we focused on evaluating the design with related objective metrics. We first present the energy characterization results and then an analysis of energy-efficiency improvement.

Since the non-Bluetooth components on the cache-watch are not power-managed, they draw about $2mA$ current all the time. Figure 7.4 presents the power consumption for the cache-watch in terms of current consumption at 3.6V when the Bluetooth adapter is in different modes. Figure 7.5 shows how the cache-watch battery lifetime changes when the average communication interval changes. If the Bluetooth adapter is not powered off in the STANDBY mode, the battery lifetime for the $800mAh$ battery pack is more than three days. If the Bluetooth adapter is powered off, the battery lifetime is more than doubled.

Most of the time, the cache-watch stays disconnected from iPAQ and there is no energy cost for Bluetooth. Assuming a typical $1KB$ data exchange each time a connection is made, the time for

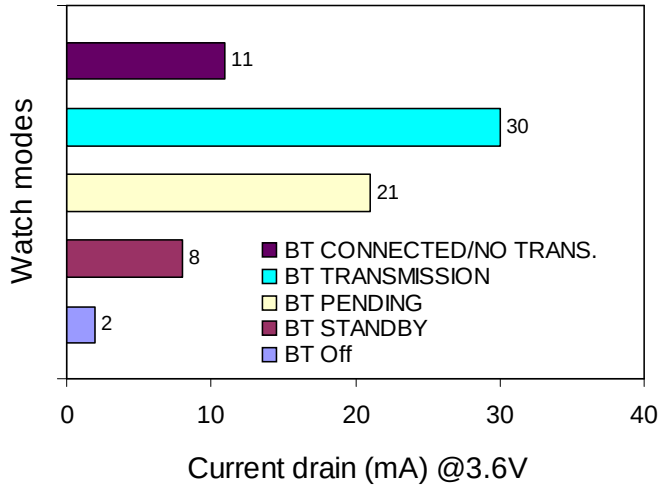


Figure 7.4: Power consumption for the cache-watch in different modes

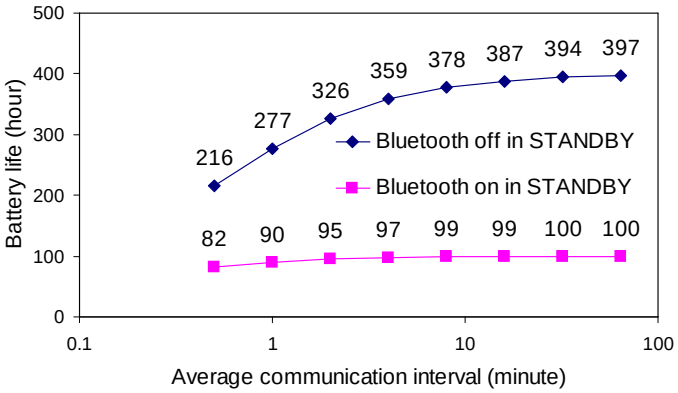


Figure 7.5: Cache-watch battery lifetime for different average communication intervals

data exchange is about 0.118 second. The corresponding energy cost for iPAQ is $84.2mJ$ (based on a power consumption of $470 + 244 = 714mW$ from Figure 5.1). Let T_p denote the time it takes the cache-watch and iPAQ to establish a connection cooperatively and T_s denote the average interval between two communication events. Note that when iPAQ is in the PENDING mode doing Paging, the power consumption is $P_{host} = (84 + 244) = 328mW$ (see Figure 5.1). On the other hand, we assume that if the cache-watch is not used, the user has to access iPAQ directly at a frequency f with an average duration of T_{access} ($f < 1/T_{access}$). The average power consumption, P_h, of iPAQ during usage, is about $(244 + 82 + 444) = 770mW$ and $(244 + 82) = 326mW$ with and without

the back light, respectively. Let Δf denote the reduction in the iPAQ access frequency f due to the use of the cache-watch. Therefore, using the cache-watch improves the energy efficiency of iPAQ if

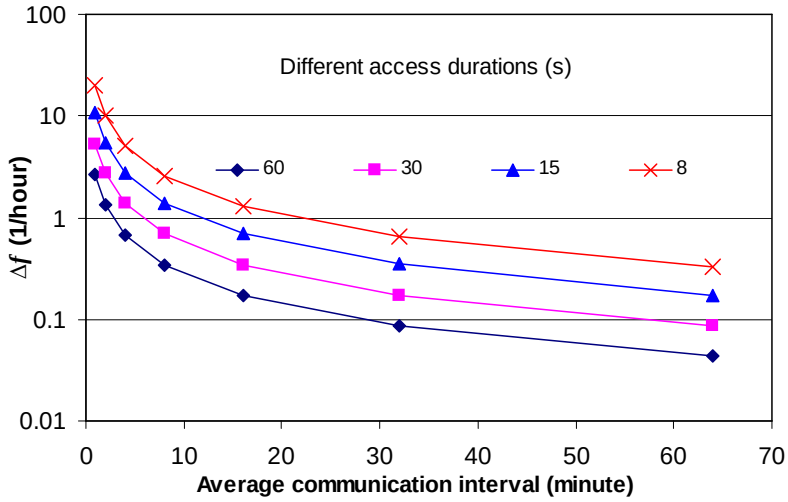
$$\frac{84.2 + P_{host} \cdot T_p}{T_s + 0.118 + T_{access}} < P_h \cdot \Delta f \cdot T_{access} \tag{7.1}$$

where the left hand side gives the extra power consumption in iPAQ due to Bluetooth activities and the right hand side gives the power reduction due to reduced iPAQ accesses. Figure 7.6 plots the minimal frequency reduction in terms of number of accesses per hour for the cache device to improve the iPAQ energy efficiency with different average communication intervals (T_s) and average access durations (T_{access}). Based on our measurements, $T_p = 2.5s$ is used. It presents the results for day-time (without the back light) and night-time (with the back light) access. Different lines represent different average access durations from 8 to 60 seconds. The figure clearly shows how many accesses per hour have to be outsourced to the cache device to improve the host computer energy efficiency. For day-time usage, if the cache communicates with the host every 30 minutes on an average, the host energy efficiency will be improved even if only two 30-second accesses or three 15-second accesses can be outsourced in a day. For night-time usage, even half the number of such outsourcings will still improve energy efficiency.

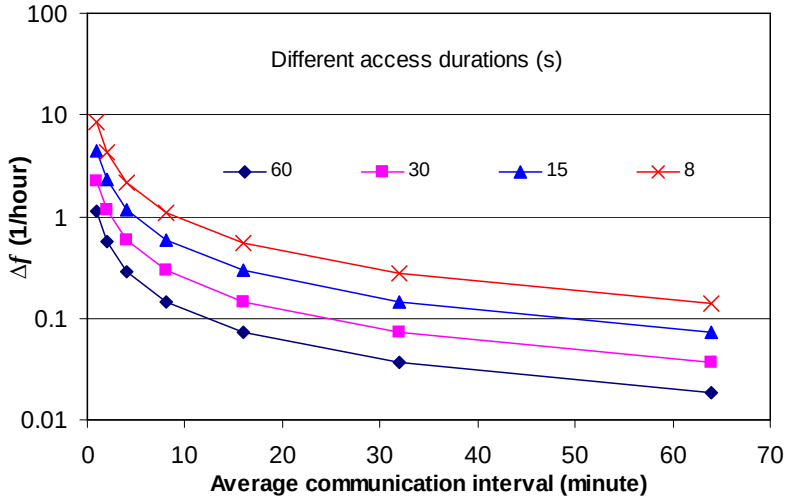
7.4 Design issues

We have shown how the cache-watch can improve the energy efficiency for the host with objective measures. The cache-watch is one example of wireless interface cache devices. In the following discussion, we highlight the important issues involved in the design of such devices.

Wireless communication: In terms of form factor, wireless communication between the host and cache devices is almost mandatory. There are several wireless personal-area network technologies intended for different data rate and application scenarios. We used Bluetooth in the current watch prototype because its availability on iPAQ and in the market facilitates prototyping. Unfortunately, Bluetooth, especially the iPAQ Bluetooth, imposes a high energy overhead for iPAQ to establish a connection with the cache-watch, as is apparent from the factor, $P_{host} \cdot T_p$ in Equation (7.1). The iPAQ Bluetooth is a software implementation, which requires the whole system,



(a) Day-time access without the back light



(b) Night-time access with the back light

Figure 7.6: Minimal frequency reduction for improving energy efficiency

except the display, to be on in order to establish a connection. Therefore, P_{host} becomes $328mW$ instead of $84mW$. This issue, however, can be resolved if the iPAQ Bluetooth is implemented in a separate hardware system. If P_{host} can be reduced to $100mW$, Δf in Figure 7.6 will be reduced by almost three times. On the other hand, it is impossible to establish a Bluetooth connection in much less than $T_p = 2.5s$. With IEEE 802.15.4 or customized radio modules with a lower power consumption, shorter connection establishment time or connectionless communication may be employed to replace Bluetooth since the required data rate is not high. This, however, requires extra hardware to be added to iPAQ.

Tasks for outsourcing: User studies will be critical for determining which tasks should be outsourced to obtain the best tradeoff between the energy efficiency of the host computer and complexity/cost of the cache device. First, instead of determining which tasks should be outsourced, the cache device designer should determine which input/output services the cache device should provide. It is important that new iPAQ applications can be developed to utilize such services, and users can specify the applications that should be outsourced to utilize them. For example, the cache-watch exports its display services through the synchronization protocol and any iPAQ application can utilize such services by specifying what to display. This gives application developers and users most flexibility. Second, even if a task itself suffers productivity degradation after being outsourced to a cache device, energy efficiency and overall productivity may still be improved. For example, browsing a text message from the cache-watch will be slower than reading it directly from the iPAQ display. However, if the overhead for the user to take out the iPAQ and power it on/off is considered, obtaining information from the cache-watch may even be faster. Another example is BlackBerry, popular smart-phones by RIM Ltd., which can host email tasks. A BlackBerry device can be viewed as an interface cache device for the laptop computer. Text entry on a BlackBerry device is based on thumbing, which is much slower than text entry with a full-size keyboard on the laptop computer. However, a study [71] funded by the manufacturer showed that with a Blackberry smart-phone, the laptop usage decreased. The primary reason is that with a more readily accessible smart-phone , a user can better utilize downtime for productivity, $e.g.$, reading/writing emails while waiting in a line.

Battery partition: A simple question for our interface cache proposal would be: what if we just give the extra battery capacity of the cache device to the host. In the prototype, such an extra battery capacity will give iPAQ an extra operating time of about two hours. The rationale behind a cache device is that we can achieve a longer system operating time by giving some battery capacity to a low-power interface cache device. Therefore, when designing an interface cache device, we must consider the usage patterns of both the cache and host devices and user's expectations of their battery lifetimes to achieve the best system operating time. Again, user studies will be extremely important.

7.5 Related devices

Reducing the cost of the interface cache device improves its energy efficiency, while increasing its hit rate makes it more useful, capable of catering to more applications. However, these two goals are contradictory. To be energy-efficient, a device must be simple, whereas to be more useful, it must be feature-rich. Our approach was to implement a minimal set of features/functionalities on the cache device. Similar wrist-worn devices have been designed including the IBM Linux Watch [97] and Microsoft SPOT watch [143]. Ericsson demonstrated years ago a Bluetooth watch, Infowear, which can be synchronized with PCs [185]. All these watches were intended to be self-contained computer systems. If viewed as cache devices for interfacing, they lie on the other extreme of the spectrum, embracing a feature-rich and power-hungry design. Indeed, the author of [72] blamed the high price and short battery lifetime for the lackluster market reception of the Microsoft SPOT watch. They differ drastically from our design of the interface cache device, which emphasizes a low-power minimalist approach. The TiltType wrist device demonstrated in [49] displays text messages from the computer using an XML-like protocol [48]. With a similar minimalist approach, it was investigated primarily as an input device based on TiltType. None of these devices was proposed to improve the energy efficiency of the host computer.

Related to the philosophy of using a low-power interface cache device, Intel's personal server project envisions that the personal server, a handheld computer-like device, utilizes wall-powered displays in the environment [215]. Since it can be much more energy-efficient to transmit user

interface specifications wirelessly than rendering and presenting the user interfaces, such a parasitic mechanism can be another user interface solution to overcome the energy efficiency bottleneck due to a slow user.

7.6 Chapter summary

This chapter tackled the slow-user problem in a very different way from Chapter 6. Instead of improving user productivity, it argued that the interfacing power consumption can be reduced to an extreme without sacrificing much user productivity. It proposed a low-power low-cost device to which a host computer outsources simple yet frequent tasks. Such a device, serving a similar goal as the cache does for memory, is called an interface cache. This chapter presented our design and prototype of a Bluetooth wrist-watch, called cache-watch, that serves as the interface cache for a mobile system. While most other digital watches are designed as complete stand-alone computer systems, the cache-watch is designed purely to serve the host computer. The system functionality is carefully partitioned so that only minimal functionality is placed on the cache-watch. Experiments and analysis presented in this chapter showed that such an interface cache will be able to improve the energy efficiency of its host computer significantly.

Chapter 8

Power Management Based on User Delay Prediction

Chapter 6 addressed the slow-user problem by improving user productivity without much power overhead. On the contrary, Chapter 7 addressed it by bringing interfacing power down to an extreme without much user productivity overhead. This chapter tackles the slow-user problem in a third way. As a mobile system spends most of its time and energy in idle periods, the Amdahl's law of energy discussed in Section 5.6 motivates us to focus on the idle periods, or user delays, during human-computer interaction for opportunities to improve energy efficiency. This chapter addresses power management based on user delay prediction. We first revisit the slow-user problem in Section 8.1. We address how an interactive application can be modeled as a state-transition diagram in Section 8.2. Based on such a modeling, we present psychology-based and history-based user delay predictions in Sections 8.3 and 8.4, respectively. We show how the proposed user prediction techniques can be implemented in Section 8.5. The predicted user delays can be combined with DPM/DVS techniques to significantly reduce energy consumption. We offer a theoretical analysis for user delay prediction-based DPM/DVS policies in Section 8.6. In Section 8.7, we describe the system power model and power management policies that utilize user delay prediction. We describe the experimental setup and present experimental results in Section 8.8. We discuss the results in Section 8.9 and conclude in Section 8.10.

8.1 The slow-user problem revisited

Most interactive applications simply block and wait idly between user inputs. Such applications are called *passively interactive*. For example, most personal information management (PIM) and productivity applications belong to this category. Some other applications proceed in their own way if there is no user input, e.g., many gaming and GPS applications. Such applications are called *actively interactive*. In this chapter, we focus on passively interactive applications and just use the term "interactive" to refer to them unless otherwise indicated.

Figure 8.1 shows the power trace of a typical interactive application, Qtopia [175] $Calculator$, on a Sharp Zaurus SL-5500 handheld computer when a user computes $(89 \times 56) \div 45$. The handheld is the same one as the Zaurus used in Chapter 4. The power is sampled 400 times per second. The energy measurement system presented in Section 4.3 is used.

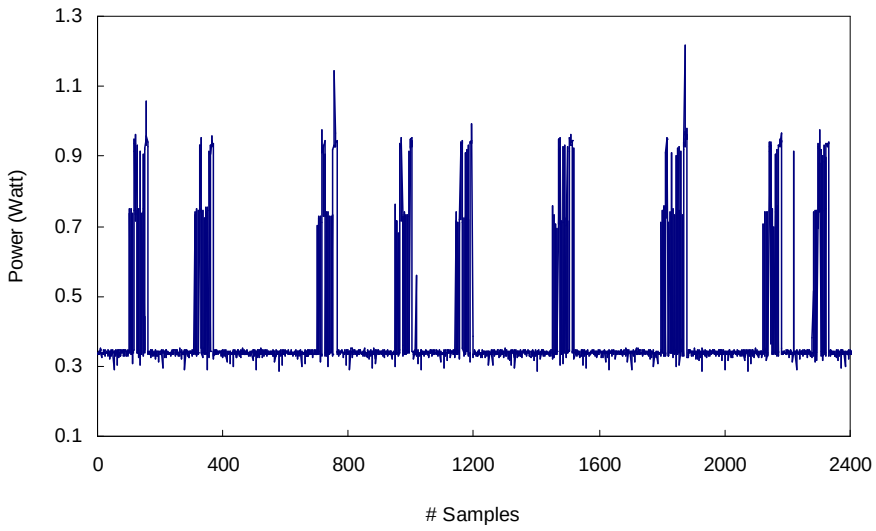


Figure 8.1: Power consumption of Sharp Zaurus while running $Calculator$.

$Calculator$ is used in a manner similar to a real calculator. Its GUI presents buttons similar to the hardware buttons on a real calculator. The user enters inputs by tapping the GUI buttons with a stylus.

As shown in Figure 8.1, there are valleys separated by nine major power peaks, which correspond to $Calculator$'s responses to tappings of GUI buttons. In the valleys, the system waits for

Table 8.1: Percentage of system time and energy spent waiting for user input

Benchmarks	User	Total time (s)	Time (%)	Energy (%)
Calculator	1	48	99.4	98.5
	2	49	99.8	97.8
Filebrowser	1	188	99.1	97.6
	2	106	98.7	96.4
Go	1	1,215	97.9	94.2
	2	259	94.6	90.2
Solitaire	1	734	99.8	99.6
	2	397	99.1	97.4

user input while the operating system (OS) does maintenance jobs like handling timer interrupts and scheduling, which introduces small fluctuations and several minor spikes in the valley. Such power characteristics are typical of most interactive systems.

To see how much time and energy the idle valleys take, we analyzed usage traces from two users for four Qtopia applications shipped with the mobile system (see Section 8.8 for details). The percentage of total time and energy the mobile system spent waiting for user input is presented in Table 8.1. Clearly, over 90% of the time and energy was spent in waiting for user input. Similar findings have been made in [53] but for desktop computers. This is exactly the slow-user problem. More importantly, most of the waiting periods are longer than 500ms. Such system idle periods are long enough for many systems to save energy by entering a low-power mode with a relatively long exit latency, which is impossible for commonly used timeout-based power management. However, if the system is woken up from the low-power mode by user input, the exit latencies may be too long to go unnoticed by users, and may lead to severe degradation in user productivity [23, 140]. Therefore, to effectively utilize such opportunities and avoid sacrifice in computer responsiveness,

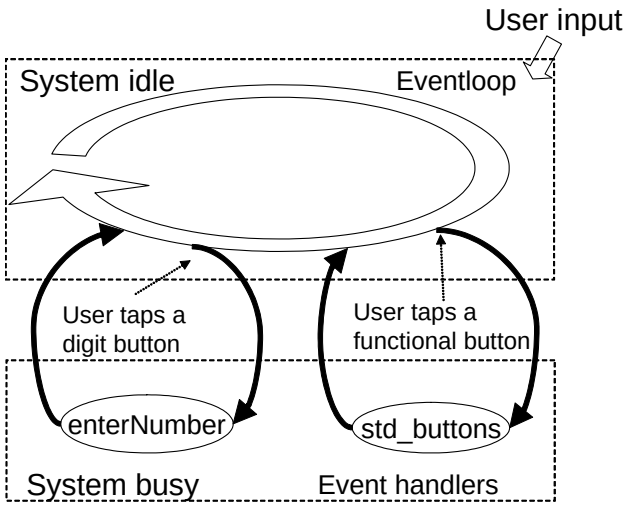


Figure 8.2: Interactive application: eventloop and event handlers

two questions must be answered:

- When does the system begin waiting for user input?
- How long does it wait?

This chapter is primarily concerned with seeking answers to these questions. We next outline our approaches.

When does the system begin waiting for user input: Most interactive applications are implemented in an event-driven fashion with an eventloop and a number of event handlers. When the system is in the eventloop, it is idle, waiting for user input. Upon receiving a user input, the system calls the corresponding event handlers to respond. After the event handlers finish execution, the system returns to the eventloop. This is illustrated in Figure 8.2 using the Qtopia $Calculator$ as an example. Note that only the two primary event handlers, *enterNumber(int n)* and *std_buttons(int n)*, are shown for clarity. They are invoked by user tapping a digit button and a functional button, respectively.

Since the system starts waiting whenever it returns to the eventloop, the application can simply notify the system when it enters the eventloop.

How long does the system wait: This question is equivalent to how long the user delay is. Without the knowledge of what the user is responding to, it is extremely difficult to answer it. This is exactly why most conventional DPM/DVS policies are not much successful for interactive systems, as mentioned in Section 2.2.4. Any information from the interactive application will help. If we know the application type, we may be able to estimate user delays better since, for example, user delays for text typing are statistically much shorter than those for playing Go. We can keep a user delay record for each application and predict its next user delay based on the application-specific record. This will improve the predictability of user delays, yet still suffer from the fact that the user may incur very different delays in different parts of the same application. For example, the user delay after clicking the "File" menu in a text editor will be much longer than that after keying in a letter.

A step further from the above monolithic view of an application would be to make a distinction between different parts of an application that have a different impact on user delays. We propose to model or partition the application from the user's perspective. Each partition, called *state* in the following discussion, presents relatively consistent information to the user, and requires relatively similar user reactions. Therefore, such a state will have more consistent and predictable user delays than the whole application.

8.2 Interaction modeling

In this section, we explain how an interactive application can be partitioned into states so that the user interface is relatively consistent in any given state, making the state user delay more predictable.

State-transition diagram extraction: State-transition diagrams (STDs) [216] have been proposed earlier to model computer-human interaction and specify user interface design. In this approach, an interactive application is modeled as a finite-state machine. Each state corresponds to a unique functional stage that waits for user input. The application changes state upon a user input/action. Such an STD, provided as a specification of the application, offers a design guide. However, we need an STD extracted from the implementation in order to implement user delay prediction.

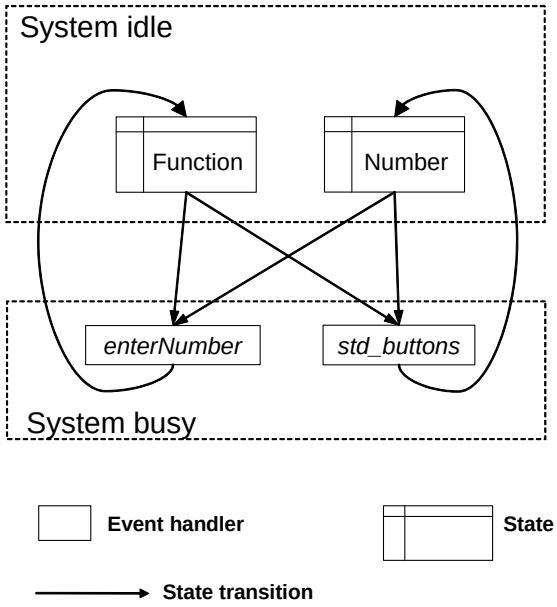


Figure 8.3: An STD for the Qtopia $Calculator$.

As illustrated in Figure 8.2, interactive applications are usually implemented with an eventloop and event handlers, which define an STD. The application waits for user input in some specific "state." Since an event handler processes user inputs and generates new information for the user, it transitions the application into another state (in some cases, the transition may be to the same state). Therefore, event handlers, with their triggering user inputs/events, correspond to transition arcs of the STD. If we assume every event handler transitions the application into a unique state, an STD is naturally defined by the event handlers. The event handlers that lead to a state in the STD are called *state triggers*. The event handlers that lead the application out of a state are called *state escapers*.

As an example, the STD for the Qtopia $Calculator$ is extracted as shown in Figure 8.3. There are two states: $Number$ after a digit button is tapped and $Function$ after a function button is tapped. The corresponding state triggers are *enterNumber(int n)* and *std_buttons(int n)*, respectively. These two event handlers are state escapers for both $Number$ and $Function$. Such a $complete$ STD is actually the most detailed STD that can be extracted from the implementation. It provides the complete system-user interaction description. Note that an event handler in the implementation may correspond to more than one transition arc since the same event handler can be activated from different states. Therefore, all the arcs entering a state correspond to the same event handler. In the

complete STD, there is a one-to-one mapping between the set of states and the set of event handlers.

If we only care about some specific user delays during system-user interaction or it is hard to derive the complete STD, we can identify event handlers which lead to the important states only. Instead of a complete STD, we will then have a $partial$ STD, in which there is a one-to-one mapping between the set of states and a subset of event handlers. Moreover, an STD, whether complete or partial, can be compacted by clustering multiple similar states into one, based on user interface content or profiling, to form a *clustered* STD. In a clustered STD, a state may correspond to more than one event handler.

The event handler-based STD provides us an opportunity to implement state-specific user delay prediction. Once we know that a set of event handlers will lead to a state, we can insert code into these event handlers to convey to the power manager the identity of the state and the predicted user delay. This is addressed in Section 8.5.

User delay prediction: Unlike many other prediction problems, errors in user delay prediction may have very different effects on the system and the user. In other words, the cost of errors is heterogeneous, making solutions from standard time-series prediction theories [17] inapplicable. If the predicted delay is shorter than the real user delay, the prediction is an underestimation and conservative. If it is longer than the real user delay, it is an overestimation and aggressive. Underestimation wastes power-saving opportunities, and overestimation may lead to user-perceptible latencies if the system is put into a power-saving mode with a long exit latency. It has been agreed upon in human-computer interaction research that a longer system response time may lead to user productivity degradation and such a degradation is dependent upon users and tasks [184]. For example, it was demonstrated in [75] that system response latencies beyond one second could degrade user productivity by about 50% for some tasks. Such a degradation may easily get rid of any gain in energy efficiency by using power-saving modes. Unfortunately, there are no established direct quantitative links between energy efficiency degradation and user-perceived latencies. In this study, we will evaluate prediction methods in terms of what tradeoffs between energy savings and user-perceptible latencies they can make. One method can be better than another only in the sense of Pareto optimality.

8.3 Psychology-based user delay models

If the content of the user interface of an STD state is known, the length of the user delay can be predicted based on what a user sees and how she or he needs to respond. There are three basic processes involved in a user's response to a system [25]. During the *perceptual* process, the user senses inputs from the physical world. In our scenario, it is the process of the user reading information presented on the display. During the *cognitive* process, the user decides on a course of action based on the input. In our scenario, it is the process of the user deciding which button to push, which menu item to click, etc. Finally, during the *motor* process, the user executes the decision by mechanical movements. These three processes are consecutive in time. Before the user physically touches the system input peripherals, the system is idle.

Perceptual delay: For most modern computer systems, users get information from the system through visual and auditory channels by reading and listening. In this study, we focus on the visual channel.

Psychological studies have shown that humans read through discrete eye movements, which consist of $fixations$ and $saccades$. Fixations are the state in which the eyes are focused on a object statically. Saccades represent rapid eye movements from one fixation location to another. Information is absorbed only in fixation, which lasts from 60 to $700ms$. A saccade takes about $30ms$. To estimate the perceptual delay, we have to estimate the number of fixations (or the number of saccades) and the duration of each. In order not to negatively impact user productivity, we estimate durations of these delays conservatively.

Computer display information can be categorized as graphical and textual. Graphical information is presented through graphical objects which consist of 1) window objects, such as buttons, radiobuttons, menus etc., and 2) pictures or figures. To simplify delay estimation, we assume each graphical object requires one separate fixation and each fixation lasts for the minimal $60ms$. Therefore, for each graphical object, a delay of $90ms$ (one minimal fixation plus one saccade) is added.

For textual information, psychological studies [26] have shown that college students can raud (read with comprehension) at a typical rate of 300 standard-length words per minute or five per second. The standard-length word was assumed to have six characters. For a text with n words, the

typical time for rauding is

$$T = \frac{n \times cpw}{Wpm \times 6} = \frac{n \times cpw}{30} \quad (s)$$

where cpw is the average number of characters per word and Wpm is the rauding rate in terms of the number of standard-length words per minute. If a text is associated with a graphical object, the reading time is used as the fixation length for the latter. For example, the delay to get information from a message box which contains a four-word text with an average of six characters per word, and two buttons with "OK" and "Cancel" on them, respectively, can be estimated as

$$T = 0.03 + \frac{4 \times 6}{30} + 0.03 + \frac{1 \times 2}{30} + 0.03 + \frac{1 \times 6}{30} = 1.157 \quad (s)$$

where there are three saccades and three fixations, whose lengths are estimated using the rauding rate.

Cognitive delay: The Hick-Hyman Law [86, 96] states that the time to make a decision (the reaction time, or RT) based on N distinct choices of equal probabilities is given by

$$RT = a + b \log_2 N \quad (s) \tag{8.1}$$

Parameters a and b are constants that can be empirically determined. Based on the information in [25], we assume a to be 0 and b to be $\frac{1}{7}$. Therefore, to decide which one from two buttons to tap, a user would approximately think for 0.14 second. To choose one item from a menu with four different items, a user would approximately think for 0.29 second. Table 8.2 summarizes how we apply the Hick-Hyman Law to various user interface features.

The Hick-Hyman Law can be useful only if the number of different choices, N, is known. Unfortunately, it is hard or even impossible to estimate N for many cognitive processes. For example, menu selection may actually involve more choices for a user than the number of menu items because the user may evaluate the consequence of selecting an item and the possible subsequent choices to decide whether to select that item or not. A good *Go* player evaluates the current choice by considering what choices he or she would have many steps after this pending move. In this study, we conservatively assume N to be the number of most direct choices.

Motor delay: It also takes time for a user to make a physical movement. In most mobile systems, users respond by touching virtual objects on the touch-screen with a stylus. The Fitts

Table 8.2: Number of choices for different user interface features used in the Hick-Hyman Law

GUI features	Number of choices
Group of N buttons	N
Group of N radiobuttons	N
Each checkbox	2 (on or off)
Menu bar with N items	N
A list of N items	N

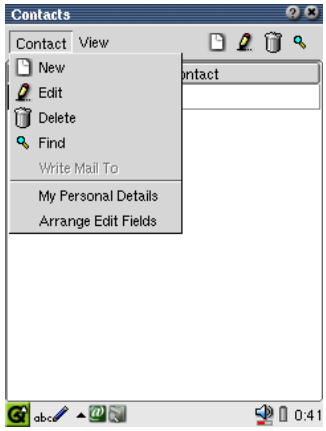


Figure 8.4: A menu window from the Qtopia application $Contact$.

Law [50, 138] states that for a normal human being, the motor time (MT) to carry out a movement is related to the size of the target and the distance to the target as

$$MT = 0.23 + 0.166 \cdot \log_2(\frac{A}{W} + 1) \quad (s) \tag{8.2}$$

where A is the amplitude of the movement and W the width of the target as measured in the direction of motion. We have adopted the values for coefficients from [138].

The above three models are together referred to as *the psychology-based model* in this chapter. As an example, there are four items on the popup menu window shown in Figure 8.4. They will require at least two fixations and two saccades. The perceptual process will take about 180ms. The

Hick-Hyman Law indicates that about 290ms will be needed for making a decision from the four choices. Suppose the hand movement altitude is about one quarter of the screen height and the width of the target (the menu item) is about one sixteenth of the screen height. Then the Fitts Law indicates that the motor time will be about 600ms. Altogether, this GUI state will take an average user more than one second to respond to. User delays of such a duration are long enough to put the system into a low-power mode with an exit latency up to several hundred milliseconds. However, they are still too short for conventional timeout-based power management policies.

Perceptual and motor delays can be predicted relatively well. However, the cognitive delay can be only conservatively predicted based on the Hick-Hyman Law, as mentioned before. Therefore, the psychology-based model is used to predict the lower bound on the user delay in this study. In the next section, we propose a history-based user delay model to predict the actual delay.

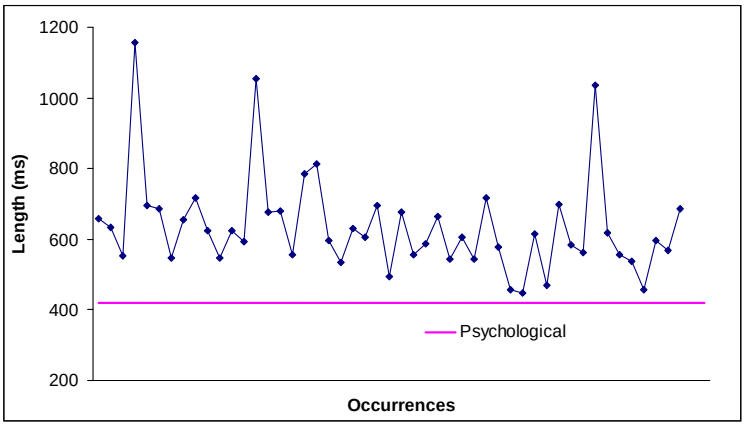
8.4 History-based user delay model

The psychology-based model is user interface content-based. It predicts user delay based on what users will see. In some cases, it may be difficult to know the user interface content. Moreover, it only offers one tradeoff between energy savings and user-perceptible latencies, and does not adapt to different users and the same user at different times. Therefore, we devise a user delay model based on past observations. Before introducing the model, we first look into the statistics of user delays.

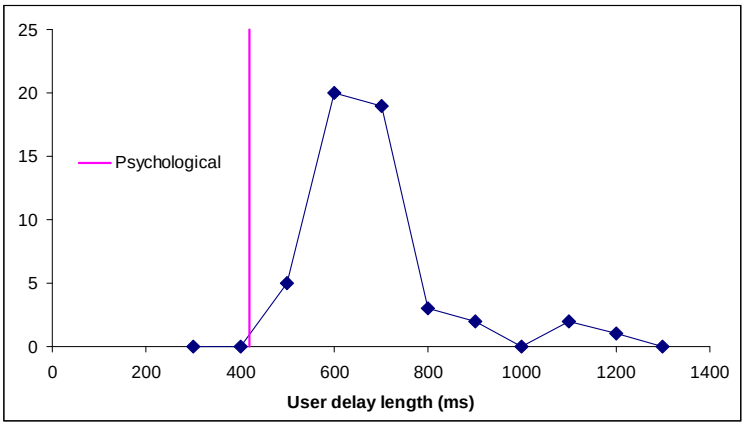
User delay statistics: Figure 8.5(a) shows the trace of user delays for the state *Number* of the Qtopia $Calculator$, collected over three days. Note that the X-axis is not time. The lower bound predicted by the psychology-based model is also plotted in both figures.

As illustrated by Figure 8.5, the user delays for a state are quite randomly but tightly centered around the mean length at which the major peak in the distribution is located. A small group of randomly occurring long delays form a minor peak in the distribution too. Such randomness makes accurate user delay prediction impossible. The trace also shows the learning effect over the three days: the delay length in general declines slightly, however, in a less obvious fashion. We also observed that the standard deviation is large if the mean is large.

History-based model: If we have gathered enough history of the user delays of a state, we



(a) User delay trace



(b) User delay length distribution

Figure 8.5: User delay statistics for State $Number$ of $Calculator$ for User 1.

can obtain the distribution similar to Figure 8.5(b). By using different percentiles of the history, we can make different tradeoffs between overestimation and underestimation, leading to different tradeoffs between energy savings and user-perceptible latencies. For example, if we use the minimal value or a value at a very low percentile in the history record, we can essentially achieve a lower-bound prediction even better than the prediction based on the psychology-based model. There are, however, some limitations of such methods. First of all, they are not reliable until a significant history record is available. For example, using the minimal value in the history record can lead to a lot of overestimation before a reasonable short user delay is observed. Second, they do not capture long-term trends such as the learning effect effectively. The tradeoff may undergo unwanted changes over time. Therefore, although such long-term history-based methods have some value, short-term

history-based methods are preferred since they do not require a significant history and can adapt to long-term trends efficiently. Moreover, short-term history-based methods will be computationally much simpler in general. Therefore, we devise a prediction method similar to the median filter based on recent history. Let D_s^i denote the ith last observed delay for an STD state s and N the total number of past user delays recorded for that state.

The delay record is sorted to generate a new series so that

$$D_s^1 < D_s^2 < ... < D_s^N$$

Then we define the *habitual set* $\Phi(p)$ as

$$\Phi(p) = \{D_s^{\lfloor N \cdot p \rfloor + 1]}, D_s^{\lfloor N \cdot p \rfloor + 2}, ..., D_s^{\lfloor N \cdot (1-p) \rfloor}\}$$

where $0 \leqslant p \leqslant 0.5$ with a typical value of 0.25. $\Phi(p)$ contains the central $N \cdot (1 - 2 \cdot p)$ items of the sorted series. It is a set extension of the concept of a median. When $p = 0.5$, it reduces to the median. When $p = 0$, it leads to the mean instead. Let m_Φ denote the mean of $\Phi(p)$. Then the next user delay for the state is predicted as

$$D = \beta \cdot m_\Phi$$

where $0 < \beta$. β is called the $pessimism$ factor and is used to generate different tradeoffs between overestimation and underestimation. There are two rationales behind using a multiplicative factor instead of an additive one. First, different states have different mean user delays; and second, states with larger means usually have larger standard deviations. A constant multiplicative factor works better for different states than a constant additive factor.

History-based vs. psychology-based models: First of all, the history-based and psychology-based models are intended for different purposes. The psychology-based model is intended to predict the lower bound of the user delay length of a state. On the other hand, the history model is to predict the true length of the next delay. By using the pessimism factor to scale the prediction, it can achieve many tradeoffs between energy savings and user-perceptible latencies. Moreover, the psychology-based model is intended to predict the lower bound as a constant that holds for all users and even after enough practice. On the other hand, the history model is intended to adapt to different users and the same users at different times effectively.

The history- and psychology-based models also complement each other. Indeed, a history-based model using the minimal or a small percentile of the history can achieve the same goal as the psychology-based model to predict the lower bound on user delay length but after enough observations have been made. It is useful only when a long history is available for the user and initial errors of overestimation can be tolerated. In the case that overestimations are not tolerable, the psychology-based model can be used before a long history is available.

The history-based model enjoys simplicity of implementation. No knowledge about the user interface content is required. On the other hand, the psychology-based model has to know the user interface content, which may be complex and dynamic. Moreover, the Hick-Hyman Law has limitations in modeling complex cognitive processes. The psychology-based model is more accurate for user interfaces that do not involve complicated cognitive processes. Therefore, different delay models should be used for different states and different applications to exploit the strengths of both the history-based and psychology-based models.

8.5 Implementation of user delay prediction

In this section, we address how user delay prediction can be implemented based on the application source code and GUI toolkit. We assume the application can convey to the power manager the predicted user delay, D, via a functional call $NotifyDelay(D)$. The power manager will then determine what step to take for saving energy, which will be addressed in Section 8.7.

Fully customized implementation: When the application source code is available, one can extract the STD for important states and augment the corresponding event handlers for user delay prediction. The beginning of a state's user delay is marked by the end of its triggers, where $NotifyDelay()$ should be inserted. For history-based prediction, a global variable, $current\_state$, is added to note the current state. It is updated at the end of state triggers. A global array of data structures is added for maintaining a delay record for each state. The array is indexed by $current\_state$.

The arrival of the next event, or the start of a state's escapers, marks the finish of a user delay. Code can be inserted into the eventloop or corresponding state escapers to estimate the real user

delay and update the state's user delay history.

In the $Calculator$ example, the start and finish of *enterNumber(int n)* and *std_buttons(int n)* mark the finish and start of user delays, respectively. At their start, the last user delay is estimated and used to update the delay record according to $current\_state$. At their finish, $current\_state$ is updated, the starting time of a user delay is marked, and $NotifyDelay()$ is inserted.

GUI toolkit-based implementation: GUI applications are usually built using GUI toolkits, which provide implementations of the eventloop mechanism, GUI objects and their corresponding event handlers.

The proposed user delay prediction mechanism can actually be incorporated into the GUI toolkit so that any application built with it will have the user delay prediction capability. In this subsection, we demonstrate how this can be implemented on the Qt GUI toolkit [173] using the example of popup menu windows like the one shown in Figure 8.4. In Qt, popup menus are instantiated as objects of the class *QPopupMenu*. *Qpopupmenu::show()* and *Qpopupmenu::hide()* are called whenever a popup menu window is shown and hidden, respectively. The beginning of the user delay is marked by the end of $show()$, into which $NotifyDelay()$ can be inserted.

For history-based prediction, the user delay can be estimated as the time elapsed between the end of $show()$ and the beginning of $hide()$ by augmenting $show()$ and $hide()$. The data structures that maintain the history can be added to the $QPopupMenu$ class. They are accessed at the end of $show()$ to make the prediction for $NotifyDelay()$ and updated at the beginning of $hide()$ with the newly observed used delay.

The *QPopupMenu* class also provides enough data through member methods for psychology-based prediction. $size()$ returns the size information for the popup window. *count()* and *text(int id)* return the number of items on the menu window and the text for the idth item, respectively. A psychology-based prediction can be obtained with such information.

8.6 Theoretical analysis of DPM/DVS

In the previous sections, we discussed how to determine when a user delay occurs and how to predict its length. We also demonstrated how user delay prediction can be implemented based on

application source code and GUI toolkit. With the $NotifyDelay(D)$ API, an interactive application can convey the predicted user delay to the power manager. We next show how user delays can be used to reduce energy consumption through straightforward DPM/DVS policies.

8.6.1 DPM and DVS techniques

DPM/DVS techniques change the system performance level to save power while trying to meet soft/hard deadlines. The performance level of an interactive system can be changed by shutting down idle components, putting the processor into different power modes, and scaling the frequency and supply voltage. Most modern processors used in mobile computing systems support performance level changes through software, e.g., Intel StrongARM and XScale processors [100]. Different performance levels have different power consumptions. Transitions among levels cause delay and extra energy consumption. This must be considered when performing DPM/DVS.

Since many GUI operations can be quite demanding, specially on mobile systems, the ideal setting would be the lowest possible performance level during user idle time and an appropriate higher level during system response.

8.6.2 Theoretical energy saving and latency overhead

We next present a theoretical analysis of the energy saving and latency overhead when DPM/DVS techniques are combined with user delay prediction.

For simplicity of exposition, we consider two performance levels for a given user delay. Let D denote the length of the user delay, P the system power consumption for the higher performance level, and P' that for the lower one. The energy consumption for the system during the user delay without DPM/DVS will be

$$E = D \cdot P$$

Perfect prediction: Suppose we can predict the user delay perfectly, change the system performance to the lower one after the system finishes responding, and change it back to the higher one right before the next user input. There is no latency overhead. Let P_{HL} and P_{LH} denote the performance level transition power, and T_{HL} and T_{LH} denote the delay for higher-to-lower and

lower-to-higher transitions, respectively. We assume that $D > (T_{HL} + T_{LH})$. Then the energy consumption for the system during the user delay with such a performance-level transition is given by

$$E_0 = (D - T_{HL} - T_{LH}) \cdot P' + T_{HL} \cdot P_{HL} + T_{LH} \cdot P_{LH}$$

The energy saving is therefore

$$\Delta E_0 = E - E_0$$

$$= D \cdot (P - P') - T_{HL}(P_{HL} - P') - T_{LH}(P_{LH} - P')$$

If we assume $P = P_{HL} = P_{LH}$, we have

$$\Delta E_0 = \{D - (T_{HL} + T_{LH})\} \cdot (P - P')$$

The energy saving ratio is calculated as

$$\rho_0 = \frac{\Delta E_0}{E} = \{1 - \frac{(T_{HL} + T_{LH})}{D}\} \cdot (1 - \frac{P'}{P}) \tag{8.3}$$

If the system changes the performance level back to the higher level upon a user input, we can obtain the energy saving through a similar analysis as

$$\rho' = \rho_0 - \frac{T_{LH}}{D} \cdot \frac{P'}{P} \tag{8.4}$$

In this scenario, the latency overhead is T_{LH}, which can be larger than the 50-100ms human-perceptible threshold.

Imperfect prediction: If the user delay models are used to predict D as D' and the system is put into the lower performance level if $D' > (T_{HL} + T_{LH})$ and put back into the higher performance level right before the predicted user delay elapses, we have

$$\rho = \begin{cases} \rho_0 - \frac{T_{LH}}{D} \cdot \frac{P'}{P}, & D' \geqslant D + T_{LH}; \\ \rho_0 - \frac{|D - D'|}{D} \cdot \frac{P'}{P}, & D \leqslant D' < D + T_{LH}; \\ \rho_0 - \frac{|D - D'|}{D} \cdot (1 - \frac{P'}{P}), & (T_{HL} + T_{LH}) \leqslant D' < D; \\ 0, & D' \leqslant (T_{HL} + T_{LH}). \end{cases} \tag{8.5}$$

The latency overheads for these four situations are T_{LH}, $|D' - D|$, 0, and 0, respectively.

It is obvious that the energy saving ratio will increase linearly if P'/P decreases. Given the two performance levels, the energy saving ratio is only dependent on the user delay and prediction error. Comparing ρ with ρ', we can see that using predicted user delay is actually more energy-efficient if T_{LH} is large compared with user delay prediction errors. Note that both ρ and ρ' can be negative, which means energy consumption may increase if performance-level transition is not properly done.

When $D' > D$, the performance level transition will be triggered by a user input, resulting in a delay between $D' - D$ to T_{LH}. This is called a *lazy error*. When T_{LH} is larger than the human-perceptible threshold, H, employing user delay prediction for DPM/DVS may introduce user-noticeable latencies, leading to what is called a *serious lazy error* (such errors are discussed in greater detail in Section 8.8.2). We need to address the percentage of serious lazy errors (PSLE) in this case. For simplicity of analysis, we assume the user delay, D, for a state has a triangle-distribution, $p(D)$, as shown in Figure 8.6. Such a distribution is a reasonable approximation to that shown in Figure 8.5(b). The mean, m, denotes how fast the user is in responding to this state, while the width, d, denotes how predictable the user is. Note that the standard deviation is $\sqrt{2/3}{\cdot}d$. If $\alpha \leqslant m$ is used to predict the user delay, the PSLE will be

$$PSLE = 100\% \cdot \int_0^{\alpha - H} p(D)dD$$

$$= \begin{cases} 0\%, & \alpha < m + H - d; \\ 50\% \cdot [1 - (\frac{m - \alpha + H}{d})^2], & m > \alpha \geqslant m + H - d. \end{cases}$$

It is clear that PSLE increases linearly as the prediction α approaches the mean m. It decreases linearly as the mean increases, demonstrating longer delays have fewer serious lazy errors. It also decreases as d decreases, showing better user delay predictability also reduces serious lazy errors.

8.7 Power models and DPM/DVS policies

Based on a predicted delay, the power manager selects an appropriate power-saving performance level for the state. The system returns to the higher performance level either after a time interval based on the predicted user delay, or just upon the next user input. Which of these two scenarios

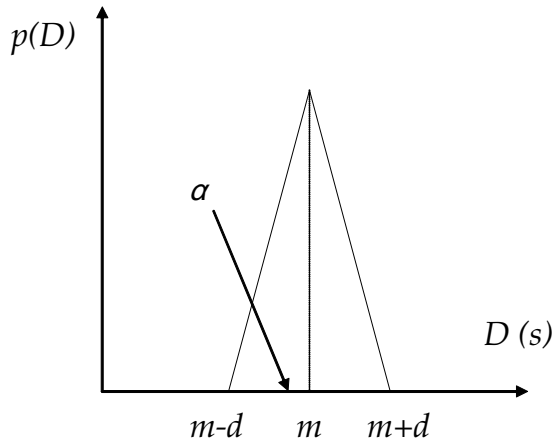


Figure 8.6: Triangle distribution of user delays for a state

is targeted depends on how aggressive dynamic power optimization needs to be. For example, the delay overhead for performance level transitions for DVS techniques is typically less than 1ms [79, 100, 170]. Thus, one could just let user inputs trigger performance level transitions. However, for more aggressive power management with an attendant increase in the delay overhead, it is often necessary to raise the performance level for prompt system response before the user input arrives. For example, the delay for an Intel StrongARM SA-1110 to transition from the SLEEP mode to the RUN mode is about 160ms [100]. In such cases, the predicted user delay can be used. In this section, we first establish a system power model for the StrongARM-based Linux handheld used in our experiments. Then we discuss possible DPM/DVS policies for it.

8.7.1 System power model

The Sharp Zaurus SL-5500 handheld has an Intel StrongARM SA-1110 processor [100], which can run in three modes: RUN, IDLE, and SLEEP. Moreover, its core clock speed can be varied between 59 and $206MHz$ in discrete steps. It is put into the IDLE mode by the Linux kernel whenever there is nothing to run. We constructed a system power model for the handheld based on power measurements and available industrial data. The modes of system operation and their corresponding power consumptions are shown in Figure 8.7. Permitted mode transitions are shown as directed arcs. The delay overhead for mode transitions is marked on the directed arcs [100].

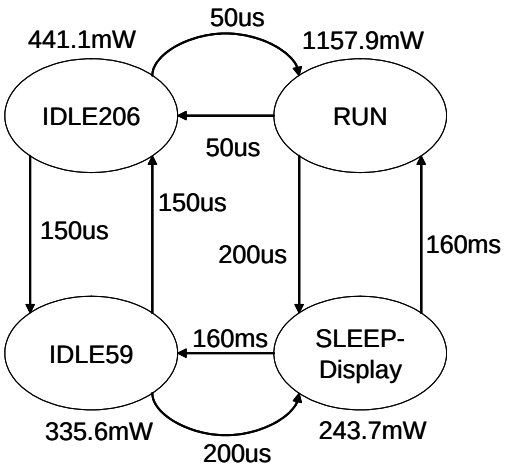


Figure 8.7: Power modes and mode transitions.

We assume that the front-light is always off and the display is always on with constant power consumption of 234.4mW. We assume the power consumption is constant for each operation mode. Notably, the display consumes about half of the system idle power.

In the system power model, RUN corresponds to the SA-1110 RUN mode when the processor is busy executing instructions. Its power is measured when the handheld repeatedly computes the DCT described in Section 4.3. IDLE206 and IDLE59 correspond to the SA-1110 IDLE mode with a core clock frequency/voltage of $206MHz/1.5V$ and $59MHz/1.25V$, respectively. IDLE59 is a hypothetical mode for the Zaurus SL-5500 based on other SA-1100 or SA-1110 systems [79, 170, 214]. It is estimated as the display power plus the measured non-display power at $206MHz$ scaled down using the same ratio of the power for IDLE59 to that for IDLE206 in the Itsy system power measurement presented in [214]. SLEEP-Display corresponds to the SA-1110 SLEEP mode when the display is left on. Therefore, its system power consumption is measured when the system is suspended, and the display power is added thereafter. Compared to IDLE59, the SLEEP-Display mode saves more power. However, it should be used with caution since the transition time back to the RUN mode is large enough for a user to notice. The power measurement equipment is the same as that described in Section 4.3.

8.7.2 DPM/DVS policies

With the user delay information as obtained using models presented in the previous sections, there are different ways to apply DPM/DVS based on the system power model. The following DPM/DVS policies are considered in this study.

Linux policy: The Linux kernel automatically puts the system into the IDLE209 mode whenever there is no process running and returns it back to the RUN mode upon interrupt. We take this as the baseline and report results for other techniques against it. The Qtopia environment is also able to power-manage the front-light and system based on timeout. However, such timeout-based policies do not benefit from user delays, which are usually no more than a couple of seconds.

Simple and lazy policies: The most straightforward DPM/DVS policy would be to put the system into the IDLE59 or the SLEEP-Display mode right after the system finishes responding to the user and put it back into the RUN mode upon a user input. These policies are called the $simple$ and $lazy$ policies, respectively. Their energy savings can be computed using Equation (8.4). Since the IDLE59 to RUN mode-transition time is small, there is no concern with regard to user-perceptible latencies for the simple policy. However, a transition from SLEEP-Display to RUN will most likely be noticed by users. The lazy policy has 100% serious lazy errors.

Perfect policy: Assuming we can predict user delays perfectly, we can choose to put the system into the SLEEP-Display mode and wake it up right before the next user input or put the system into the IDLE59 mode. The energy saving can be computed using Equation (8.3). This is called the $perfect$ policy since it gives the upper bound on energy savings based on the system power model.

Prediction policies: We can also use delay prediction, discussed in the previous sections, for DPM/DVS. The energy saving for such prediction policies can be computed using Equation (8.5). However, there are two concerns with respect to prediction errors. First, in the case of overestimation, the system will wake up from the SLEEP-Display mode upon a user input and enter the RUN mode directly, resulting in a lazy error, which may be serious. Second, in the case of underestimation, the system wakes up and transfers to the IDLE59 mode after the predicted delay to get ready for the user input and will not be able to fully exploit the idle time to reduce energy by remaining in the SLEEP-Display mode. Therefore, we report the average user delay underestimation

error. We refer to the DPM/DVS policies based on user delay prediction by the history-based and psychology-based models as $history$ and $psychological$, respectively.

Suppose in the GUI shown in Figure 8.4, it actually takes a user 1200ms to physically touch a menu item with a stylus. According to the system power model, the Linux Kernel DPM policy will consume about 0.53 Joule of system energy. Using the equations from Section 8.6.2, the simple, perfect and lazy policies will reduce it by about 24%, 38%, and 32%, respectively. The prediction policies will reduce the energy up to 38%, depending on the prediction error. If the prediction error is below 25%, the energy reduction will be between 28% and 38%. In view of the 100% serious lazy errors incurred by the lazy policy, the prediction policies are better.

8.7.3 System implementation issues

When a system is put into a energy-saving mode like SLEEP, the processor is stopped, unable to accomplish any regular OS maintenance tasks such as *kswapd* and *kupdated*. There are two solutions to this problem. The first solution is to treat any kernel event as a scheduled user input. The system determines whether and how long to stay in the SLEEP-Display mode based on the next scheduled kernel event and the predicted user delay. This is possible since most Linux kernel events are scheduled with intervals on the order of seconds. The second solution is to delay non-urgent kernel event handling until the next user input or till a certain period of time has elapsed. Another issue is that many passive interactive applications may have still maintain certain trivial activities when waiting for user input such as a blinking cursor. Activities with a short period are not compatible with the policies using the SLEEP-Display mode previously addressed. However, activities with a reasonably long period can be handled as a scheduled user input. For example, if the cursor blinks once per second, there is still enough time for the SLEEP-Display mode to save energy. To focus on the investigation of user delay prediction, we ignored these issues in this chapter by using experiments based on replaying pre-collected usage traces through the system power model.

8.8 Experiments

In this section, we describe the benchmark applications, usage traces and power measurement setup used in this study, and present the experiment results.

8.8.1 Experimental setup

An application environment, called Qtopia [175] for Linux-based handheld systems, is shipped with the Sharp Zaurus SL-5500 handheld. The source code for its applications is freely available under the General Public License. We consider four applications from Qtopia version 1.5.0 as our benchmarks. They are $Calculator$, $Filebrowser$, Go, and $Solitaire$. $Calculator$ is as described in Section 8.1. $Filebrowser$ is a GUI application for listing files in the local storage. Go and $Solitaire$ are two popular games well known by their names.

Based on an analysis of their source code, STDs were extracted. The STD for $Calculator$ is the same as that shown in Figure 8.3. The STD for $Filebrowser$ has five states after simplification: $Menu$ after the user chooses one item from any menu, $ItemClicked$ after the user taps an item in the file list, $Arrow$ after the user taps an arrow on the menu bar (to go back or go upward one level in the directory hierarchy), Dir after the user taps "Dir" on the menu bar (it waits for the user to select an item from the "Dir" menu), and $Sort$ after the user taps "Sort" on the menu bar (it waits for the user to select an item from the "Sort" menu). However, only $ItemClicked$ and $Arrow$ frequently appeared in the usage traces collected. The STDs for both Go and $Solitaire$ only have a single state, which waits for the user's next step: placing a stone or moving a card. The parameter values for user delay models were derived for the perceptual, cognitive and motor processes. Since psychology-based prediction is intended to offer the lower bound for user delays, the parameter values were chosen conservatively. They are given in Table 8.3. The predicted delays (Column 6) in the table are used as the initial values for history-based prediction. The statistics of actual user delays for states with significant occurrences in the collected traces is given in Table 8.4. The statistics show that user delays with larger means typically have larger standard deviations. Since $Calculator$ requires little cognitive processing, its psychology-based prediction is close to the mean and the standard deviations are relatively small.

Table 8.3: Parameter values for the psychology-based model

Benchmark	State	Per. (# of fix.)	Cog. (N)	Mot. (A/W)	Predicted delay (ms)
Calculator	Number	1	1	0.5	420
	Function	1	2	1	630
Filebrowser	Menu	4	4	2	1,140
	Arrow	2	2	2	820
	ItemClicked	2	4	2	960
	Dir	1	4	1	770
	Sort	1	5	1	810
Go	Main	1	1	0.5	420
Solitaire	Main	2	2	1	630

Usage trace collection: We collected system-user interaction traces for real usage of the benchmarks on the handheld. We inserted $gettimeofday()$ at the beginning and end of the event handlers (usually $slots$ in Qt [173]) in the benchmark applications to record the time in microseconds. This records when the system finishes its response and when it starts processing an incoming event and changes state. The state name is also recorded. This method counts the delay of the system in generating an event (called $signal$ in Qt) after receiving a user input as part of the user delay instead of the system busy time. Based on our experience, the error thus introduced is negligible.

Two users participated in the collection of traces. One is a male graduate student majoring in Engineering while the other is a female graduate student majoring in Social Sciences. Both are veteran computer users. Before trace collection, they were already familiar with real calculators and filebrowsing applications. The male user was also familiar with the $Patience$ game in $Solitaire$.

Table 8.4: Actual user delay statistics

Benchmark	State	User 1		User 2	
		Mean (ms)	Stdev. (ms)	Mean (ms)	Stdev. (ms)
Calculator	Number	625	186	610	263
	Function	1,026	522	640	392
Filebrowser	Menu	-	-	-	-
	Arrow	2,838	1,826	2,716	3,134
	ItemClicked	2,271	1,482	3,295	2,846
	Dir	-	-	-	-
	Sort	-	-	-	-
Go	Main	5,375	5,929	1,415	1,130
Solitaire	Main	3,040	2,152	2,150	1,887

The female user was not. Neither of them knew how to play Go. They were taught the rules for playing the games and can be regarded as beginners. They were given instructions on how to operate the handheld and use the benchmark applications. They were also given time to play with the benchmark applications on the handheld to get acquainted. Then they were asked to complete the tasks as described in Table 8.5. Each user was asked to perform the tasks once per day and for a total of three days. The total length of time of traces for each benchmark and user is given in Table 8.1. The traces were collected in an office environment. Disturbance was reasonably minimized. The users were told to perform the tasks in the same way that they would do in real life. The mean and standard deviation for each STD state's user delay are shown in Table 8.4

Trace replay: We replayed the collected traces through the system power model detailed in Section 8.7.1. We applied the power management policies described in Section 8.7.2 in different

Table 8.5: Tasks for usage trace collection

Benchmark	Task
Calculator	Compute formula, each with three two-digit numbers and two operations
Filebrowser	Find a given file on the local storage
Go	Play a new game for several minutes
Solitaire	Play a new Patience game for several minutes

replays to gauge their performance. Since the traces were pre-collected, we implicitly assumed that user behavior remained unchanged after the users encountered perceptible latencies. Therefore, the energy savings obtained by policies that incur user-perceptible latencies are only ideal, and are likely to be reduced in reality due to productivity degradation. However, since the relationship between productivity degradation and user-perceptible latencies is different in different applications, such an assumption make our evaluation more general. Moreover, since we evaluate prediction methods using tradeoffs between such ideal energy savings and user-perceptible latencies, we believe that there is enough information for system designers to make tradeoffs based on the nature of human-computer interaction in their applications.

8.8.2 Experimental results

In this section, we present experimental results using the setup described in the previous section.

Energy savings: The energy consumption is obtained by running the usage traces through the system power model with different DPM/DVS policies. In Figure 8.8, we give energy savings against the baseline (Linux policy). These include the simple, lazy, perfect, and psychology/history-based prediction policies. In the history-based prediction policy, the pessimism factor β is assumed to be 0.4 and the last seven delay records are used starting with the psychological prediction. The percentage of lazy error (PLE) (the percentage of all predictions that are overestimations) and PSLE are also reported in Figure 8.9 for the psychological (solid lines) and history (dashed lines) policies.

We adopt $50ms$ as the human perceptual threshold.

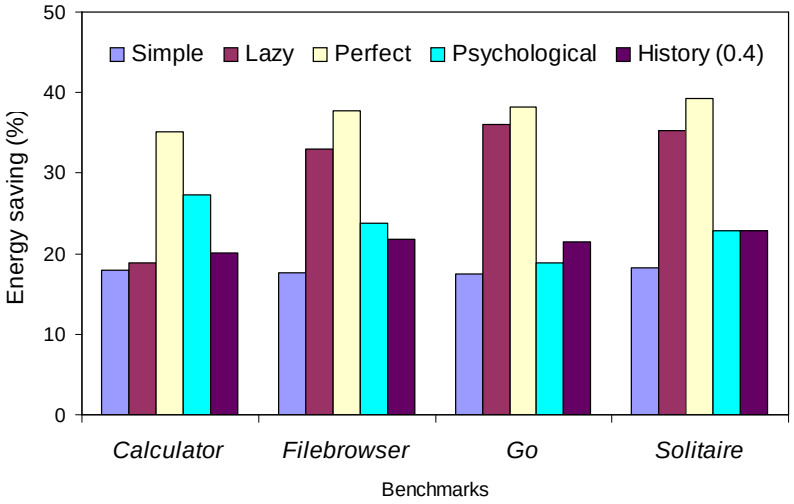
Notably, the lazy policy's energy savings are closest to the perfect policy's in three of the four benchmarks. Only in $Calculator$, it is less energy-efficient than the other policies due to the fact that user delays for $Calculator$ tend to be much shorter and predictable (see Section 8.6.2). However, the lazy policy has 100% serious lazy errors. Therefore, it will be useful only when system latencies are tolerable and the user delays are relatively long.

It is worth noting that the psychological policy performs better for $Calculator$ and $Filebrowser$ than Go and $Solitaire$ compared to the history policy. This is due to the fact that operating $Calculator$ and $Filebrowser$ is cognitively much simpler and their cognitive processes are better modeled by the Hick-Hyman Law.

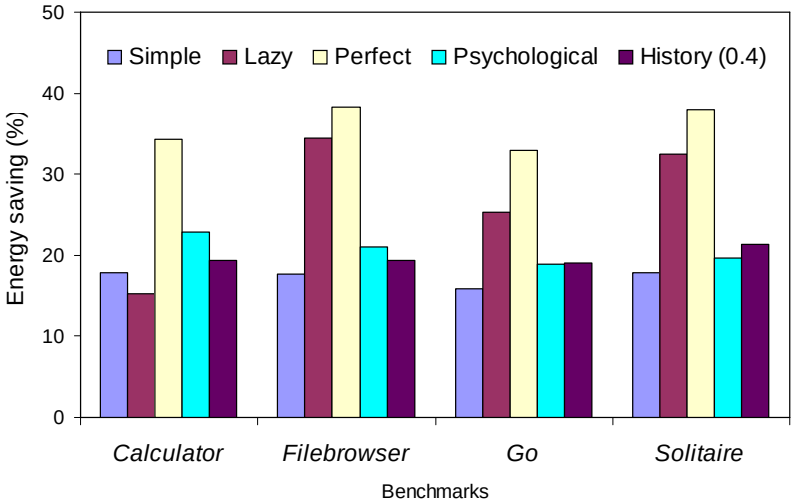
Tradeoff between lazy errors and energy savings: For the psychological policy, the lazy errors are very few since this model is pessimistic. However, for the history policy, the pessimism factor controls the tradeoff between the lazy errors and energy saving. Figures 8.10 and 8.11 show how the energy saving changes with the percentage of lazy errors for $Calculator$ and $Filebrowser$ as the pessimism factor varies from 0.2 to 1.3 for the history policy. It also shows the tradeoff point for the psychological policy. To achieve the same energy saving as the psychological policy, the history policy needs to make a lot more lazy errors. This demonstrates the superiority of the psychological policy when avoiding lazy errors is important. These figures show that psychology-based prediction works better for $Calculator$ than $Filebrowser$ compared to history-based prediction since operating the former is cognitively much simpler.

State-awareness: To show the benefit of being state-aware in history-based prediction, Figures 8.10 and 8.11 also show the tradeoff curves for state-unaware history-based prediction, which uses the last seven observed delays of the application to predict the next delay for the same application without knowing its states. It is application-based instead of state-based. It is clear that state-based prediction is better in general. Moreover, the STDs of $Calculator$ and $Filebrowser$ are relatively simple. We expect the advantage of state-awareness to be larger for more complicated applications with a larger number of STD states.

We also note that state-awareness does not benefit User 2 for $Calculator$, meaning the user did

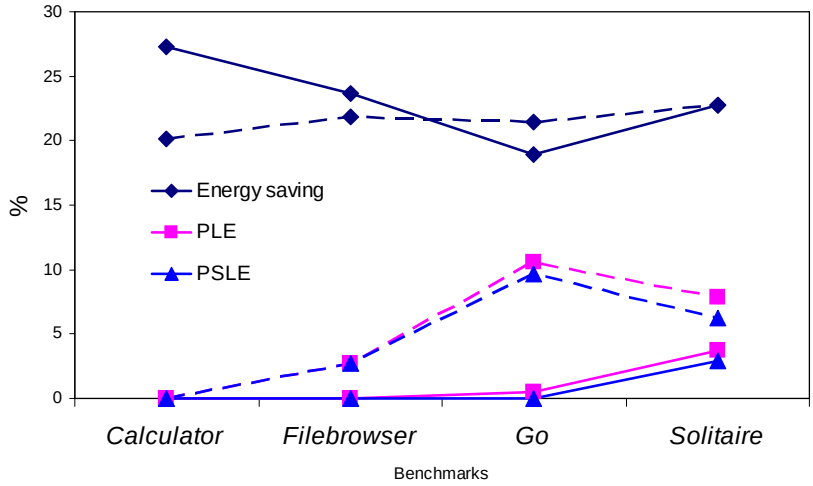


(a) User 1

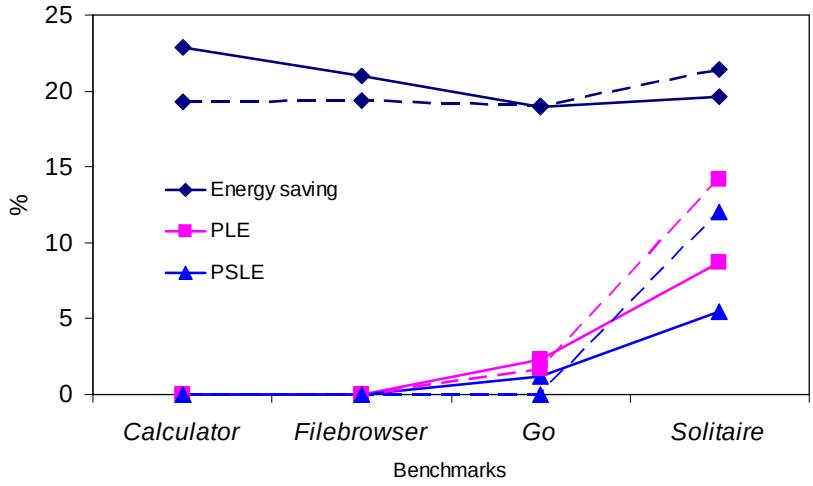


(b) User 2

Figure 8.8: Energy savings based on predicted user delays



(a) User 1



(b) User 2

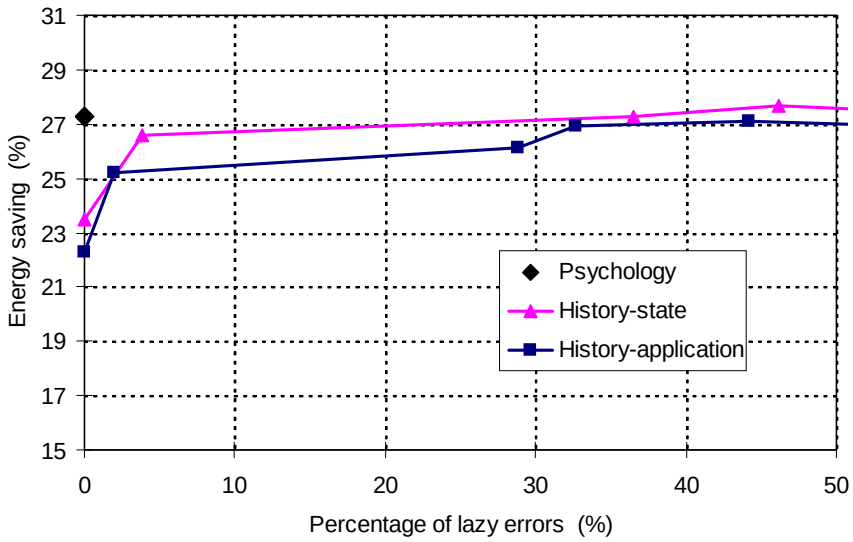
Figure 8.9: Percentage of lazy errors and energy savings

not behave very differently when responding to the two STD states. This can also be seen from Table 8.4. While the mean user delays for the $Calculator$ states in User 1's trace differ a lot, those in User 2's trace do not. This illustrates the difference in behavior between different individuals when interacting with a computer.

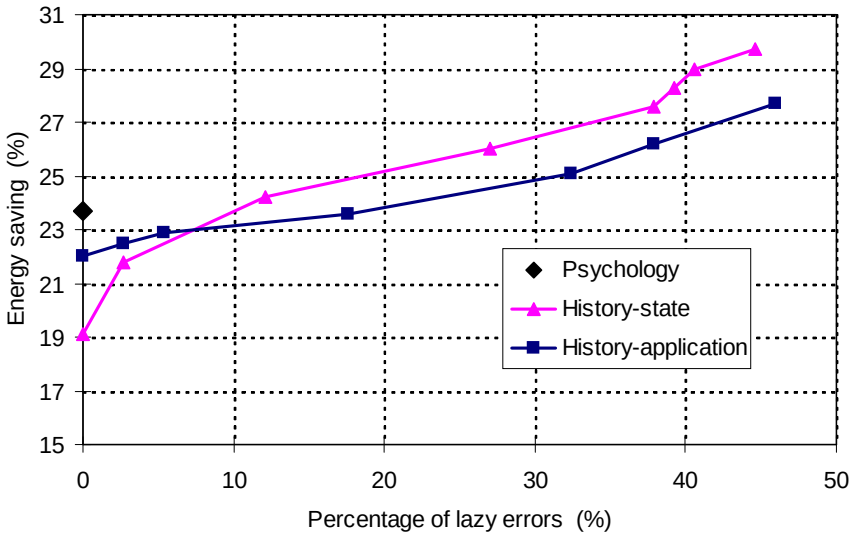
Power-saving mode – power, transition time and policy: It is obvious from the theoretical analysis in Section 8.6.2 that the energy saving will be more when the power of the power-saving mode decreases, given that all other aspects remain unchanged. However, as shown by the system model in Figure 8.7, a power-saving mode with lower power usually incurs a longer transition time back to the RUN mode. A longer transition not only increases energy overhead but may also cause a user-perceptible latency if it is larger than the human perceptual threshold, 50ms. First, we would like to investigate how the power-saving mode's power and transition time will impact energy savings. Figures 8.12 and 8.13 plot the curves of energy saving vs. the SLEEP-Display mode power for different policies (Perfect, Lazy, and History) and two different transition times (160ms and 40ms), all for User 1.

For all policies, energy savings increase linearly as the SLEEP-Display mode power decreases for all benchmarks. For $Calculator$, the user delays are relatively short, which leads to a relatively large energy saving increase when the transition time decreases from 160ms to 40ms. For other benchmarks, such a decrease does not bring much energy benefit. This is because $Calculator$ has relatively short user delays. Therefore, decreasing transition time below the human perceptual threshold is worthwhile only when the expected user delays are short. On the other hand, decreasing power-saving mode's power is always effective. For example, if there are two power saving modes with power/transition time being 150mW/45ms and 200mW/1ms, respectively, the first one will save much more energy than the second for most interactive applications without causing serious lazy errors.

The lazy policy's advantage in energy saving over the history policy increases as the power decreases. When the transition time is smaller than the human-perceptible threshold of 50ms, the lazy policy can be safely used without causing user-perceptible latencies.

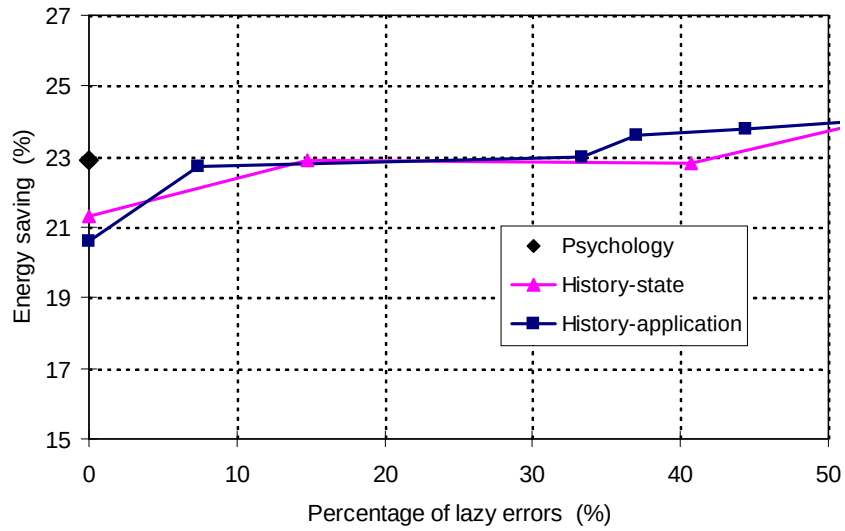


(a) *Go*

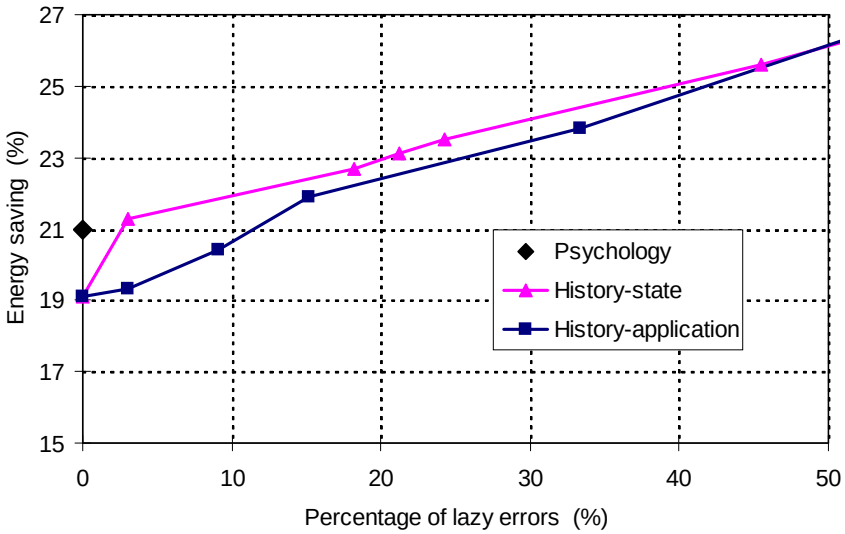


(b) *Filebrowser*

Figure 8.10: Tradeoff between percentage of lazy errors and energy savings for User 1.

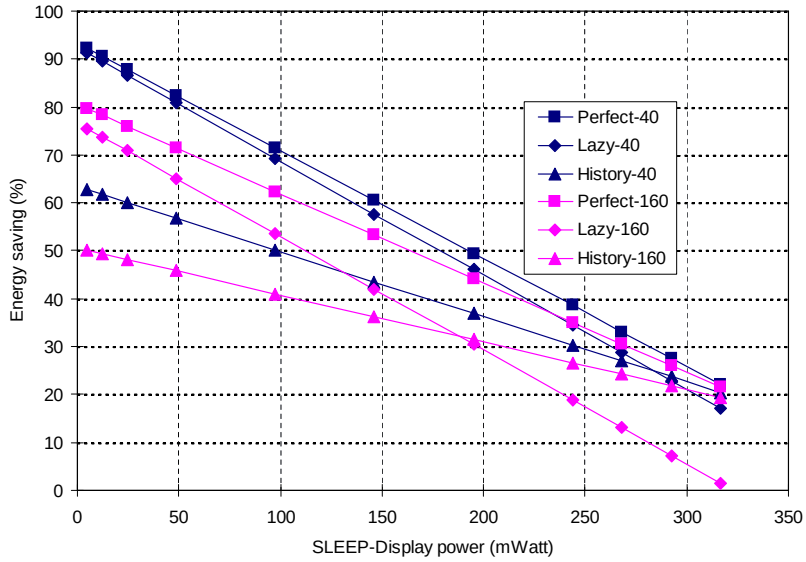


(a) *Go*

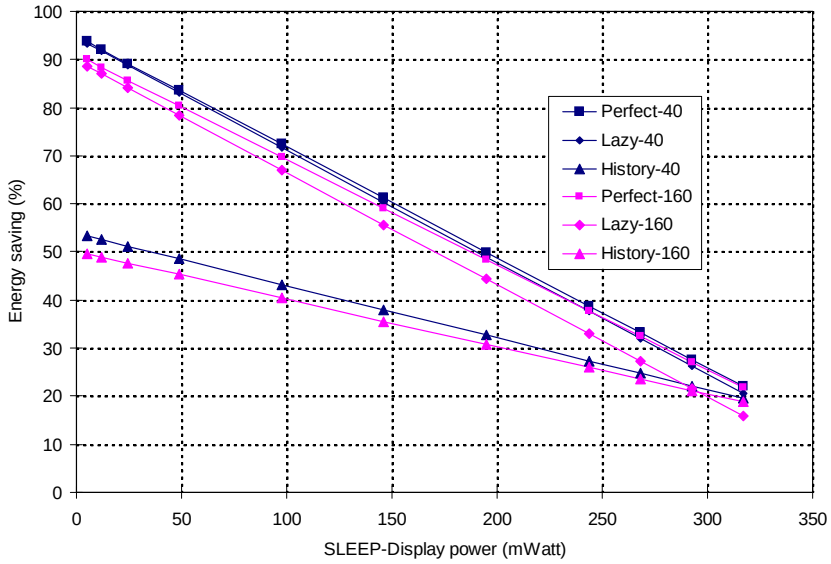


(b) *Filebrowser*

Figure 8.11: Tradeoff between percentage of lazy errors and energy savings for User 2.

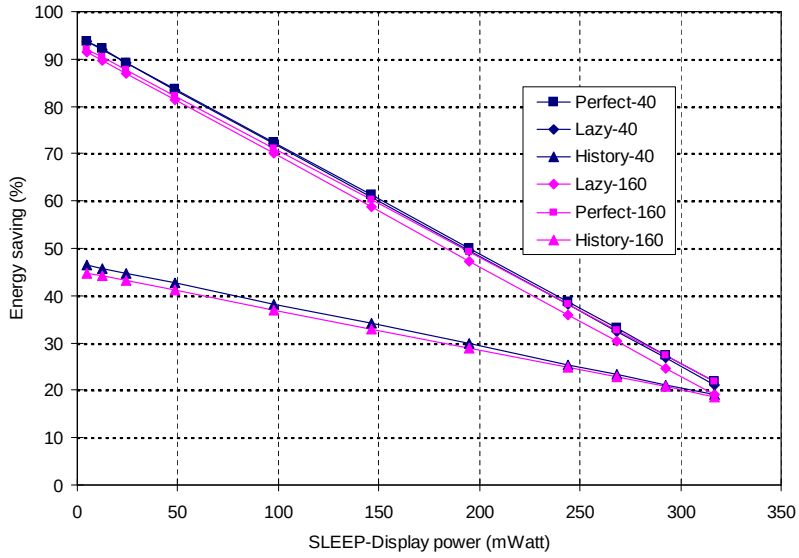


(a) *Calculator*

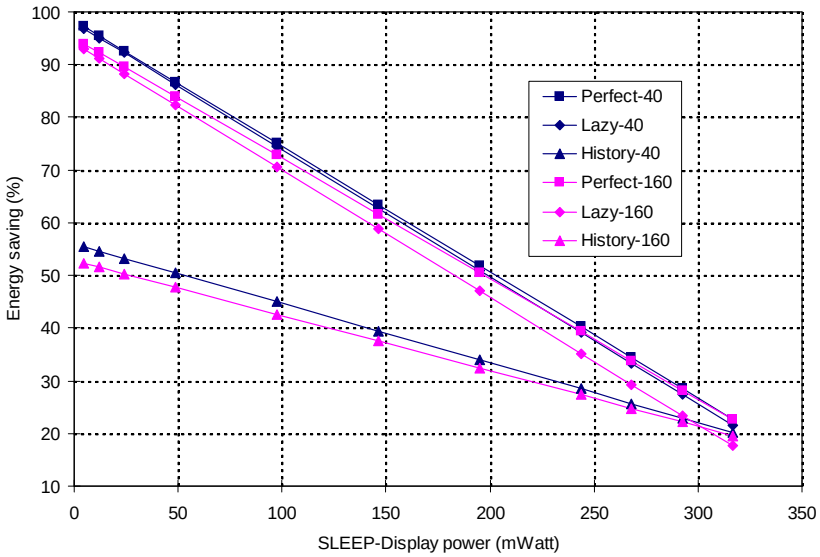


(b) *Filebrowser*

Figure 8.12: Energy saving changes with mode power and transition time



(a) *Go*



(b) *Solitaire*

Figure 8.13: Energy saving changes with mode power and transition time

8.9 Discussions

In the previous section, we showed the effectiveness of combining DPM/DVS with user delay prediction. We discuss several related issues next.

STD extraction: As may be apparent to the reader, how an STD is extracted for an interactive application is important for user delay prediction. For example, if $Calculator$ is modeled with only one state, which may seem natural, the user delay prediction will degrade as demonstrated for the history-based model for User 1. Also, the not-so-good performance of user delay models for Go and $Solitaire$ can be partially attributed to modeling them with a single-state STD. Adding states into the STD has the potential to improve intra-state consistency and the tradeoff between energy savings and lazy errors.

User delay uncertainty: Since an STD state presents relatively consistent information to the user and requires relatively consistent user reaction, we expect the corresponding user delays to be more predictable. However, there are two factors that contribute to the uncertainty in user delays, as discussed next.

First, the presented information may vary for different occurrences of the same state. For example, it can be dynamic in the case of file browsers and text readers. Such intra-state inconsistency can be reduced by using more states. The dynamic information problem can be partially solved by obtaining information about the content, such as obtaining the number of items in the current directory and the size of the text to be shown. The GUI toolkit-based implementation takes care of this readily.

Second, a state may require complicated user reactions, or permit very different user reactions. For a complicated state, the user may respond only after reading different subsets of the information presented in a state. For example, a $Solitaire$ player may not examine all stacks of cards before responding. In other words, such a state represents multiple subtasks to a user. However, the uncertainty is alleviated by the fact that there is usually one subtask most often performed by the user. For example, a $Filebrowser$ user most likely will read the file list and click a file from the list; a $Word$ user will type letters most of the time. Therefore, the most frequently-performed subtask(s) can be used to build the psychology model, as we did for Go and $Solitaire$. The history-based

prediction policy automatically benefits from this fact.

Improving prediction accuracy: We showed in Section 8.6.2 that the energy saving ratio is directly related to user delay prediction accuracy. The experimental results also demonstrated that more energy can be saved when delay can be predicted more accurately. To better utilize user delays for energy reduction, improving prediction accuracy will be important.

In this chapter, we used the psychology-based models and values of their coefficients from previous work, which were not based on mobile systems. Since our evaluation is based on a mobile system, this modeling discrepancy may prevent it from benefiting more from our methodology. We expect psychological experiments on humans using mobile systems to generate more mobile system-specific user delay models. Moreover, as we have pointed out, the cognitive delay models based on the Hick-Hyman Law may contribute most to prediction errors for the psychology-based model. Better cognitive delay models are likely to increase energy savings. Moreover, users may differ from each other in their perceptual, cognitive, and motor delays, as we can see from Table 8.4. The same user may improve his/her usage skills due to the learning process. The environment or context may also change user behavior. Although the history-based model automatically adapts, the psychology-based model does not. It would be interesting to see how adaptation of user delays to changing human behavior would improve prediction accuracy and increase energy savings.

Hardware support for interactive systems: Since the display has to be always on during user-computer interaction, changing the operation mode of other components of the system should be independent of the display. For example, the system should be able to put its processor to SLEEP without affecting its display. To the best of our knowledge, there is no such hardware support in commercially-available mobile computing systems. For displays that need refreshing, the frame-buffer and the control unit transferring data from it to the display must be on too. In most systems, these components can be on with others power-managed. For example, in the StrongARM/PXA family systems-on-chip, the framebuffer sits in the main memory and the LCD controller is one of the peripheral units. They can be active while many other units are power-managed [100].

Another solution is to use a display that maintains the content itself. In the μSleep work [16], an LCD with an integrated framebuffer was used. When the system was put into the SLEEP mode, the

LCD displayed an image in its own framebuffer automatically without support from the SA-1100 LCD controller. Moreover, bistable LCDs [114, 156, 226] can maintain the screen content even without the power supply. However, they usually incur more time and energy overheads for screen changes. They will be ideal for passive interactive systems that have relatively long user delays.

Other ways to benefit from user delay prediction: We showed how user delay prediction can be utilized for DPM/DVS in the scenario in which there is only one application under consideration. In other scenarios, an interactive application may be used with other applications running in the background. A user may enjoy music from the MP3 player, while downloading another MP3 file and playing $Solitaire$ at the same time. Therefore, it would be better if $Solitaire$ notifies the OS about predicted user delays and the OS globally schedules all the processes. The OS scheduler can make use of the predicted user delays to allocate time slots and performance levels to different processes in an energy-efficient fashion.

8.10 Chapter summary

This chapter addressed the slow-user problem in a third way: by reducing the energy consumed during user delays. It proposed an application-based user delay prediction framework. In this framework, an STD is first obtained for the application to model the interaction between the system and user. User delays are then predicted for each STD state. Within this framework, two prediction models were proposed. The history-based model predicts the actual delay based on recent observations. Since it may overestimate the delay, it should only be used when performance level transition time is not a large concern. In our experiments, it resulted in an average energy saving of 20.7% with a relatively small percentage of serious lazy errors. This chapter also showed that exploiting STD states yields a better tradeoff between lazy errors and energy savings. The psychology-based model exploits the user interface information further and predicts the lower bound on user delays, i.e., how long it takes the user to read, decide, and move. Our experiments showed that an average of 21.9% energy savings can be obtained with negligible serious lazy errors. The chapter showed that the tradeoff between lazy errors and energy savings achieved by the psychology-based model is beyond the capacity of the history-based model. It also demonstrated how DVS can be simply

combined with user delay prediction. An average of 17.6% system energy reduction can be easily achieved without introducing any user-noticeable delays. For applications more tolerant of system delays and with longer user delays, an average of 28.9% system energy reduction can be achieved.

Chapter 9

Conclusions

We conclude the dissertation in this chapter. We first summarize it in Section 9.1, then discuss its limitations in Section 9.2, and address possible future directions for mobile system design from the user's perspective in Section 9.3.

9.1 Dissertation summary

This dissertation is devoted to the slow-user problem of mobile systems. That is, an increasingly fast and power-hungry computer spends most of its time and energy in waiting for a constantly slow human user for interactive tasks. This problem was highlighted by Chapters 3 through 5 using theoretical analysis and experimental characterization.

The slow-user problem motivates us to address the energy efficiency issue from the user's perspective and evaluate it using the product of user productivity and power efficiency. In this dissertation, we presented three different approaches to tackle the problem from the user's perspective. The approach described in Chapter 6 was to boost user productivity without incurring too much power overhead. We present the design and prototype of a Bluetooth-based PAN of low-power interfacing devices. The PAN is intended to provide a user natural and energy-efficient access to the computing capacity on a mobile system. The approach described in Chapter 7 was to bring the interfacing power down to an extreme without sacrificing user productivity much. We presented the concept of an interface cache and a prototype design, cache-watch. The cache-watch hosts simple, yet fre-

quent, interactive tasks from a mobile system to save the latter's battery lifetime. The approach described in Chapter 8 was to reduce power consumption in the idle periods of a mobile system during human-computer interaction. We proposed modeling of an interactive application with a state-transition diagram, and prediction of user delays using psychological theories and history. We then aggressively power manage the mobile system in idle periods based on the user delay prediction. The user's perspective and these three approaches represent a novel interdisciplinary endeavor to tackle the slow-user problem and improve the energy efficiency of mobile systems.

9.2 Limitations

We next describe the limitations of our techniques.

Limitations of the work: The work included in this dissertation is the very first effort to tackle the energy efficiency issue from the user's perspective. Although we emphasize an interdisciplinary approach that combines knowledge from both computer engineering and human-computer interaction, we have to admit that we are not human-computer interaction researchers. As a result, work in this dissertation is limited in its strength in terms of user studies, as is apparent in Chapters 6 through 8. Specifically, we have not yet quantitatively evaluated the user productivity improvement in Chapter 6, conducted user studies of the cache-watch in Chapter 7, or quantitatively evaluated the user productivity impact of serious lazy errors in Chapter 8. We have to leave these works for the future, and seek help from human-computer interaction researchers. However, we believe that this dissertation is complete as a computer engineering thesis that borrows knowledge from the field of human-computer interaction.

Limitations of the approaches: One of most important design principles of a complex system is to separate concerns. That is, a designer should focus on a limited set of concerns, and a system module should encapsulate its implementation details while providing well-defined interfaces, the fewer, the better. Unfortunately, the approaches from the user's perspective require system designers to consider user productivity, and user interface designers to consider battery lifetime. They also require system modules to interact with each other more, e.g., user interfaces to interact with the battery. These approaches improve the energy efficiency of a mobile system only with more

information about the user and user interfaces. In this sense, they are less generic as compared to lower level low-power techniques, which save power without knowledge beyond the hardware. Nevertheless, we believe that such an issue is likely to be alleviated after future research generates solid guidelines for designers to address the new concern, and tools that automate the interaction between user interfaces and power management.

9.3 Future work

As this dissertation is the very first effort to address energy efficiency from the user's perspective, we expect more research to follow. Especially, we expect human-computer interaction researchers to join us to quantitatively evaluate user productivity and conduct user studies for energy-efficient mobile system design. To them, the message from this dissertation is clear: energy efficiency must be considered as a direct consequence of user productivity; power consumption must be taken into consideration when evaluating a user interface.

Moreover, we expect more computer engineering research on mobile systems, especially smartphones, for higher user productivity and more services. As illustrated in Chapters 6 and 7, we need to examine new technologies and investigate their impact on human-computer interaction. While human-computer interaction researchers focus on user studies of a new interfacing device, we, as computer engineering researchers, need to invent new devices enabled by new technologies. We call this *technology-driven interface design*.

Contrary to design from the user's perspective, past computer engineering research focuses on design from the perspective of computing. Because of Moore's Law, the cost of computing keeps on dropping while the cost of human labor does not. As many researchers have argued [60, 123, 192, 193], it is time that we rethink computing from the user's perspective. Instead of high-performance computing, we need to target high-productivity computing. User productivity should no longer be the concern of user interface designers only; it should become the concern of computer engineering researchers as well. We call this *user-centric computing design*. This dissertation is just a humble step in this direction.

References

[1] A. Acquaviva, L. Benini, and B. Riccó. *Compilers and Operating Systems for Low Power*, Energy characterization of embedded real-time operating systems, pages 53–73. Kluwer Academic Publishers, Norwell, MA, 2003.

[2] Advanced Configuration & Power Interface. *http://www.acpi.info*.

[3] A. V. Aho, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 1986.

[4] R. Allen and K. Kennedy. *Optimizing Compilers for Modern Architectures: A Dependence-based Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, 2001.

[5] M. Anand, E. B. Nightingale, and J. Flinn. Ghosts in the machine: Interfaces for better power management. In *Proc. USENIX/ACM Int. Conf. Mobile Systems, Applications, & Services*, pages 23–35, June 2004.

[6] D. F. Bacon, S. L. Graham, and O. J. Sharp. Compiler transformations for high-performance computing. *ACM Computing Surveys*, 26(4):345–420, Dec. 1994.

[7] R. W. Bailey. *Human Performance Engineering: Design High Quality Professional User Interfaces for Computer Products, Applications and Systems*. Prentice Hall PTR, Upper Saddle River, NJ, 3rd edition, 1996.

[8] K. Barr and K. Asanovic. Energy aware lossless data compression. In *Proc. USENIX/ACM Int. Conf. Mobile Systems, Applications, & Services*, pages 231–244, May 2003.

[9] 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 *Proc. ACM Int. Conf. Compilers, Architecture, and Synthesis for Embedded Systems*, pages 203–210, Nov. 2001.

[10] L. Benini, A. Bogliolo, G. A. Paleologo, and G. De Micheli. Policy optimization for dynamic power management. *IEEE Trans. Computer-Aided Design of IC & Systems*, 18(6):813–833, June 1999.

[11] L. Benini and G. De Micheli. *Dynamic Power Management: Design Techniques and CAD Tools*. Kluwer Academic Publishers, Norwell, MA, 1998.

[12] D. Bertozzi, L. Benini, and B. Ricco. Power aware network interface management for streaming multimedia. In *Proc. IEEE Wireless Communnication Network Conf.*, pages 926–930, Orlando, FL, Mar. 2002.

[13] V. Bharghavan and V. Gupta. A framework for application adaptation in mobile computing environments. In *Proc. IEEE Computer Software and Application Conf.*, pages 573–579, Nov. 1997.

[14] 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 *Proc. Int. Conf. Human Computer Interaction with Mobile Devices & Services*, pages 13–24, Sept. 2004.

[15] Bluetooth. *http://www.bluetooth.org/*.

[16] L. S. Brakmo, D. A. Wallach, and M. A. Viredaz. μSleep: A technique for reducing energy consumption in handheld devices. In *Proc. USENIX/ACM Int. Conf. Mobile Systems, Applications, & Services*, pages 12–22, June 2004.

[17] P. Brockwell and R. A. Davis. *Introduction to Time Series and Forecasting*. Springer-Verlag, New York, 1996.

[18] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *Proc. Int. Symp. Computer Architecture*, pages 83–94, June 2000.

[19] BTAccess library. *http://www.high-point.com*.

[20] T. D. Burd and R. W. Brodersen. Design issues for dynamic voltage scaling. In *Proc. Int. Symp. Low Power Electronics & Design*, pages 9–14, July 2000.

[21] T. D. Burd, T. A. Pering, A. J. Stratakos, and R. W. Brodersen. A dynamic voltage scaled microprocessor system. *IEEE J. Solid-State Circuit*, 35(11):1571–1580, Nov. 2000.

[22] P. Buser and M. Imbert. *Vision*. The MIT Press, Cambridge, MA, 1992.

[23] T. W. Butler. Computer response time and user performance. In *Proc. Conf. Human Factors in Computing Systems*, pages 58–62, Dec. 1983.

[24] L. Cai and Y.-H. Lu. Energy management using buffer memory for streaming data. *IEEE Trans. Computer-Aided Design of IC & Systems*, 24(2):141–152, Feb. 2005.

[25] S. K. Card, T. P. Moran, and A. Newell. *The Psychology of Human-Computer Interaction*. Lawrence Erlbaum Assoc., Hillsdale, NJ, 1983.

[26] R. P. Carver. *Reading Rate: A Review of Research and Theory*. Academic Press, Inc., San Diego, CA, 1990.

[27] A. P. Chandrakasan and R. W. Brodersen, editors. *Low-Power CMOS Design*. Wiley-IEEE Press, 1997.

[28] W.-C. Cheng and M. Pedram. Power minimization in a backlit TFT-LCD display by concurrent brightness and contrast scaling. *IEEE Trans. Consumer Electronics*, 50(1):25–32, Feb. 2004.

[29] I. Choi, H. Shim, and N. Chang. Low-power color TFT LCD display for handheld embedded systems. In *Proc. Int. Symp. Low Power Electronics & Design*, pages 112–117, Aug. 2002.

[30] K. Choi, K. Dantu, W.-C. Cheng, and M. Pedram. Frame-based dynamic voltage and frequency scaling for a MPEG decoder. In *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, pages 732–737, Nov. 2002.

[31] D. Chung. Mobile platform display technology advancements. *Intel Developer Forum*, Sept. 2002.

[32] E.-Y. Chung, L. Benini, A. Bogliolo, Y.-H. Lu, and G. De Micheli. Power management for nonstationary service requests. *IEEE Trans. Computers*, 51(11):1345–1361, Nov. 2002.

[33] E.-Y. Chung, G. De Micheli, and L. Benini. Contents provider-assisted dynamic voltage scaling for low energy multimedia applications. In *Proc. Int. Symp. Low Power Electronics & Design*, pages 42–47, Aug. 2002.

[34] T. L. Cignetti, K. Komarov, and C. S. Ellis. Energy estimation tools for the Palm. In *Proc. ACM Int. Wkshp. Modeling, Analysis and Simulation of Wireless and Mobile Systems*, pages 96–103, Aug. 2000.

[35] P. M. Commarford and J. R. Lewis. Models of throughput rates for dictation and voice spelling for handheld devices. *Int. J. Speech Technology*, 7(1):69–79, Jan. 2004.

[36] K. D. Cooper, D. Subramanian, and L. Torczon. Adaptive optimizing compilers for the 21st century. *J. Supercomputing*, 23(1):7–22, May 2001.

[37] A. B. Dalton and C. S. Ellis. Sensing user intention and context for energy management. In *Proc. Wkshp. Hot Topics in Operating Systems*, pages 23–35, May 2003.

[38] R. P. Dick, G. Lakshminarayana, A. Raghunathan, and N. K. Jha. Power analysis of embedded operating systems. In *Proc. ACM/IEEE Design Automation Conf.*, pages 312–315, June 2000.

[39] F. Douglis, P. Krishnan, and B. N. Bershad. Adaptive disk spin-down policies for mobile computers. In *Proc. 2nd USENIX Symp. Mobile & Location-Independent Computing*, pages 121–137, Apr. 1995.

[40] E-Ink: Electronic Ink. *http://www.eink.com*.

[41] Editor's Picks: PDA stories from around the Web. Is the PDA dead? *http://news.com.com/2009-1025_3-5224632.html*, Aug. 2004.

[42] C. Efstratiou, A. Friday, N. Davies, and K. Cheverst. A platform supporting coordinated adaptation in mobile systems. In *Proc. IEEE Wkshp. Mobile Computing Systems & Applications*, pages 128–137, June 2002.

[43] C. S. Ellis. The case for higher-level power management. In *Proc. Wkshp. Hot Topics in Operating Systems*, pages 162–167, Mar. 1999.

[44] Eyal De Lara, D. S. Wallach, and W. Zwaenepol. Puppeteer: component-based adaptation for mobile computing. In *Proc. USENIX Symp. Internet Technologies & Systems*, pages 159–170, Mar. 2001.

[45] K. I. Farkas, J. Flinn, G. Back, D. Grunwald, and J. Anderson. Quantifying the energy consumption of a pocket computer and a Java virtual machine. In *Proc. ACM Int. Conf. Measurement & Modeling of Computer Systems*, pages 252–263, June 2000.

[46] Fastap. *http://www.digitwireless.com/*.

[47] Y. Fei, L. Zhong, and N. K. Jha. An energy-aware framework for coordinated dynamic software management in mobile computers. In *Proc. IEEE/ACM Int. Symp. Modeling, Analysis & Simulation of Computer and Telecommunications Systems*, pages 306–317, Oct. 2004.

[48] K. P. Fishkin. Personal communication. Dec. 2004.

[49] K. P. Fishkin, K. Partridge, and S. Chatterjee. User interface components for lightweight WPANs. *IEEE Pervasive Computing Magazine*, (4), Oct.-Dec. 2002.

[50] P. M. Fitts. The information capacity of human motor system in controlling the amplitude of movement. *J. Experimental Psychology*, (47):381–391, 1954.

[51] K. Flautner and T. Mudge. Vertigo: automatic performance-setting for Linux. In *Proc. Symp. Operating Systems Design & Implementation*, pages 105–116, Dec. 2002.

[52] K. Flautner, S. Reinhardt, and T. Mudge. Automatic performance setting for dynamic voltage scaling. In *Proc. ACM Ann. Int. Conf. Mobile Computing & Networking*, pages 260–271, July 2001.

[53] K. Flautner, R. Uhlig, S. Reinhardt, and T. Mudge. Thread-level parallelism and interactive performance of desktop applications. In *Proc. Int. Conf. Architectural Support for Programming Languages & Operating Systems*, pages 129–138, Aug. 2000.

[54] M. Fleischmann. Dynamic power management for Crusoe^{TM} processors. *White Paper, Transmeta Corporation*, Jan. 2001.

[55] J. Flinn, K. I. Farkas, and J. Anderson. Power and energy characterization of the Itsy pocket computer (version 1.5). Technical Report Technical Note TN-56, Compaq Western Research Laboratory, Feb. 2000.

[56] J. Flinn and M. Satyanarayanan. Energy-aware adaptation for mobile applications. In *Proc. ACM Symp. Operating Systems Principles*, pages 48–63, Dec. 1999.

[57] J. Flinn and M. Satyanarayanan. Managing battery lifetime with energy-aware adaptation. *ACM Trans. Computer Systems*, 22(2):137–179, May 2004.

[58] S. Forrest. The roadmap to high efficiency organic light emitting devices. *Organic Electronics*, 4(2-3):45–48, Sept. 2003.

[59] G. H. Gelinck *et al*. Flexible active-matrix electronic ink display. *Nature*, 3:106–110, Feb. 2004.

[60] D. Garlan, D. Siewiorek, A. Smailagic, and P. Steenkiste. Project Aura: Toward distraction-free pervasive computing. *IEEE Pervasive Computing*, 1(2):22–31, Apr. 2002.

[61] F. Gatti, A. Acquaviva, L. Benini, and B. Riccó. Low power control techniques for TFT LCD displays. In *Proc. ACM Int. Conf. Compilers, Architecture, and Synthesis for Embedded Systems*, pages 218–224, Oct. 2002.

[62] W. W. Gaver. Auditory icons: Using sound in computer interfaces. *Human-Computer Interaction*, 2:167–177, 1986.

[63] S. A. Gelfand. *Hearing: An Introduction to Psychological and Physiological Acoustics*. Marcel Dekker, Inc, New York, NY, 3rd edition, 1998.

[64] J. Gemmell, G. Bell, R. Lueder, S. Drucker, and C. Wong. MyLifeBits: Fulfilling the Memex vision. In *Proc. ACM Int. Conf. Multimedia*, pages 235–238, Nov. 2002.

[65] J. Gemmell, L. Williams, K. Wood, R. Lueder, and G. Bell. Passive capture and ensuing issues for a personal lifetime store. In *Proc. ACM Wkshp. Continuous Archival & Retrieval of Personal Experiences*, pages 48–55, Oct. 2004.

[66] S. Ghiasi and D. Grunwald. A comparison of two architectural power models. In *Proc. Int. Wkshp. Power-Aware Computer Systems*, pages 137–152, Nov. 2000.

[67] GIMP toolkit. *http://www.gtk.org*.

[68] D. Goldberg and A. Goodisman. Stylus user interfaces for manipulating text. In *Proc. ACM Ann. Symp. User Interface Software & Technology*, pages 127–135, Nov. 1991.

[69] R. Golding, P. Bosch, and J. Wilkes. Idleness is not sloth. In *Proc. USENIX Winter Tech. Conf.*, pages 201–212, Jan. 1995.

[70] R. Golding, P. Bosch, and J. Wilkes. Idleness is not sloth. Tech. Rep. HPL-96-140, HP Labs, 1996.

[71] Goldman Sachs Global Equity Research. *Goldman Sachs Mobile Device Usage Study*. 2001.

[72] H. Goldstein. A dog named SPOT. *IEEE Spectrum*, 41(1):72–73, Jan. 2004.

[73] C. S. Golvin and P. Jackson. When will the phone and PDA merge? Today—But the appeal of these smartphones will be limited. Trend, Forrester Research, Dec. 2004.

[74] R. Gonzalez, B. M. Gordon, and M. A. Horowitz. Supply and threshold voltage scaling for low power CMOS. *IEEE J. Solid-State Circuits*, 32(8):1210–1216, Aug. 1997.

[75] T. Goodman and R. Spence. The effect of system response time on interactive computer aided problem solving. In *Proc. 5th Annual Conf. Computer Graphics & Interactive Techniques*, pages 100–104, Aug. 1978.

[76] K. Govil, E. Chan, and H. Wasserman. Comparing algorithm for dynamic speed-setting of a low-power CPU. In *Proc. ACM Ann. Int. Conf. Mobile Computing & Networking*, pages 13–25, Nov. 1995.

[77] GPE Palmtop environment. *http://gpe.handhelds.org*.

[78] M. Graciarena, H. F. H., K. Sonmez, and H. H. Bratt. Combining standard and throat microphones for robust speech recognition. *IEEE Signal Processing Letters*, 10(3):72–74, Mar. 2003.

[79] D. Grunwald, P. Levis, K. I. Farkas, C. B. Morrey III, and M. Neufeld. Policies for dynamic clock scheduling. In *Proc. Symp. Operating Systems Design & Implementation*, pages 73–86, Oct. 2000.

[80] S. Gurumurthi, A. Sivasubramanium, M. Kandemir, and H. Franke. DRPM: Dynamic speed control for power management in server class disks. In *Proc. Int. Symp. Computer Architecture*, pages 169–179, June 2003.

[81] Handheld market free fall continues. *http://news.com.com/2102-1047_3-5560453.html*. Feb. 2005.

[82] T. Harter, S. Vroegindeweij, E. Geelhoed, M. Manahan, and P. Ranganathan. Energy-aware user interfaces: An evaluation of user acceptance. In *Proc. Conf. Human Factors in Computing Systems*, pages 199–206, Apr. 2004.

[83] T. Heath, E. Pinheiro, J. Hom, U. Kremer, and R. Bianchini. Application transformations for energy and performance-aware device management. In *Proc. Int. Conf. Parallel Architecture & Compilation Techniques*, pages 121–131, Sept. 2002.

[84] D. P. Helmbold, D. D. E. Long, and B. Sherrod. A dynamic disk spin-down technique for mobile computing. In *Proc. ACM Ann. Int. Conf. Mobile Computing & Networking*, pages 130–142, Nov. 1996.

[85] J. L. Hennessy, D. A. Patterson, and D. Goldberg. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 3rd edition, 2002.

[86] W. E. Hick. On the rate of gain of information. *Quarterly J. Experimental Psychology*, (4):11–36, 1952.

[87] Hitachi Global Storage Technologies. *http://www.hgst.com/hdd/micro/overvw.htm*. 2003.

[88] D. A. Hodges, R. Saleh, and H. G. Jackson. *Analysis and Design of Digital Integrated Circuits in Deep Submicron Technology*. McGraw Hill, 3rd edition, July 2003.

[89] I. Hong, D. Kirovski, G. Qu, M. Potkonjak, and M. B. Srivastava. Power optimization of variable voltage core-based systems. In *Proc. ACM/IEEE Design Automation Conf.*, pages 176–181, June 1998.

[90] I. Hong, M. Potkonjak, and M. B. Srivastava. On-line scheduling of hard real-time tasks on variable voltage processor. In *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, pages 653–656, Nov. 1998.

[91] C. Hsu, U. Kremer, and M. Hsiao. Compiler-directed dynamic frequency and voltage scaling. In *Proc. Wkshp. Power-Aware Computer Systems*, Nov. 2000.

[92] C.-H. Hsu and U. Kremer. The design, implementation, and evaluation of a compiler algorithm for CPU energy reduction. In *Proc. ACM SIGPLAN Conf. Programming Languages Design & Implementation*, pages 38–48, June 2003.

[93] C. J. Hughes, J. Srinivasan, and S. V. Adve. Saving energy with architectural and frequency adaptations for multimedia applications. In *Proc. Ann. IEEE/ACM Int. Sym. Microarchitecture*, pages 250–261, Dec. 2001.

[94] D. Husemann, C. Narayanaswa, and M. Nidd. Personal mobile hub. In *Proc. IEEE Int. Symp. Wearable Computers*, pages 85–91, Oct.-Nov. 2004.

[95] C.-H. Hwang and A. C.-H. Wu. A predictive system shutdown method for energy saving of event-driven computation. *ACM Trans. Design Automation of Electronic Systems*, 5(2):226–241, Apr. 2000.

[96] R. Hyman. Stimulus information as a determinant of reaction time. *J. Experimental Psychology*, (45):188–196, 1953.

[97] IBM Linux watch. *http://www.research.ibm.com/WearableComputing*.

[98] C. Im, S. Ha, and H. Kim. Dynamic voltage scheduling with buffers in low-power multimedia applications. *ACM Trans. Embedded Computing Systems*, 3(4):686–705, Nov. 2004.

[99] Initium Promi-ESD^{TM}. *http://www.initium.co.kr/english/promi-esd.html*.

[100] Intel PCA processors manuals and guides. *http://www.intel.com/design/pca/applicationsprocessors/manuals/index.htm*.

[101] Intel QuickStart Technology. *http://www.intel.com/cd/products/services/EMEA/ENG/notebook/processors/114499.htm#technical2*.

[102] Intel SpeedStep® Technology. *http://support.intel.com/support/processors/mobile/pentiumiii/sb/CS-007509.htm*.

[103] Intel Technology In New Mobile Pentium® III Processors Turbo-Charges Mobile PCs. *http://www.intel.com/pressroom/archive/releases/mp011800.htm*. Jan. 2000.

[104] International Telecommunication Union. Broadband is key to the information society. Special report, Global Symposium for Regulators, 2005.

[105] S. Irani, S. Shukla, and R. Gupta. Online strategies for dynamic power management in systems with multiple power-saving states. *ACM Trans. Embedded Computing Systems*, 2(3):325–346, July 2003.

[106] T. Ishihara and H. Yasuura. Voltage scheduling problem for dynamically variable voltage processors. In *Proc. Int. Symp. Low Power Electronics & Design*, pages 197–202, Aug. 1998.

[107] S. Iyer, L. Luo, R. Mayo, and P. Ranganathan. Energy-adaptive display system designs for future mobile environments. In *Proc. USENIX/ACM Int. Conf. Mobile Systems, Applications, & Services*, pages 245–258, May 2003.

[108] N. K. Jha. Low power system scheduling and synthesis. In *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, pages 259–263, Nov. 2001.

[109] J. Johnson. *GUI Bloopers: Don'ts and Do's for Software Developers and Web Designers*. Morgan Kaufmann, San Francisco, CA, 2000.

[110] N. Kamijoh, T. Inoue, C. M. Olsen, M. T. Raghunath, and C. Narayanaswami. Energy trade-offs in the IBM wristwatch computer. In *Proc. IEEE Int. Symp. Wearable Computers*, pages 133–140, Oct. 2001.

[111] J. T. Kao, M. Miyazaki, and A. P. Chandrakasan. A 175-mV multiply-accumulate unit using an adaptive supply voltage and body bias architecture. *IEEE J. Solid-State Circuits*, 37(11):1545–1534, Nov. 2002.

[112] C.-M. Karat, C. Halverson, D. Horn, and J. Karat. Patterns of entry and correction in large vocabulary continuous speech recognition systems. In *Proc. Conf. Human Factors in Computing Systems*, pages 568–575, May 1999.

[113] H. Kawamoto. The history of liquid-crystal displays. *Proc. IEEE*, 90(4):460–500, Apr. 2002.

[114] Kent Displays, Inc. *http://www.kentdisplays.com/product/modules.htm*.

[115] J. Kimmel, J. Hautanten, and T. Levola. Display technologies for portable communication devices. *Proc. IEEE*, 90(4):581–590, Apr. 2002.

[116] J. Kolinski, R. Chary, A. Henroid, and B. Press. *Building the Power Efficient PC: A Developer's Guide to ACPI Power Management*. Intel Press, Aug. 2001.

[117] C. M. Krishna and Y.-H. Lee. Voltage-clock-scaling adaptive scheduling techniques for low power in hard real-time systems. *IEEE Trans. Computers*, 52(12):1586–1593, Dec. 2003.

[118] P. Krishnan, P. Long, and J. Vitter. Adaptive disk spindown via optimal rent-to-buy in probabilistic environments. *Algorithmica*, 23(1):31–56, Jan. 1999.

[119] T. Kuroda, T. Fujita, S. Mita, T. Nagamatsu, S. Yoshioka, K. Suzuki, F. Sano, M. Norishima, M. Murota, M. Kako, M. Kinugawa, M. Kakumu, and T. Sakurai. A 0.9-V, 150-MHz, 10-mW, 4mm^2 2-D discrete cosine transform core processor with variable threshold-voltage (VT) scheme. *IEEE J. Solid-State Circuits*, 31(11):1770–1779, Nov. 1996.

[120] W.-C. Kwon and T. Kim. Optimal voltage allocation techniques for dynamically variable voltage processors. *ACM Trans. Embedded Computing Systems*, 4(1):211–230, Feb. 2005.

[121] L. Deng *et al.* Distributed speech processing in MiPads multimodal user interface. *IEEE Trans. Speech & Audio Processing*, 10(8):605–619, Nov. 2002.

[122] R. Landauer. Irreversibility and heat generation in the computing process. *IBM J. Research & Development*, 3:183–191, July 1961.

[123] T. K. Landauer. *The Trouble with Computers: Usefulness, Usability, and Productivity*. The MIT Press, June 1996.

[124] S. Lee and T. Sakurai. Run-time voltage hopping for low-power real-time systems. In *Proc. ACM/IEEE Design Automation Conf.*, pages 806–809, June 2000.

[125] K. Li, R. Kumpf, P. Horton, and T. E. Anderson. A quantitative analysis of disk drive power management in portable computers. In *Proc. USENIX Winter Tech. Conf.*, pages 279–291, Jan. 1994.

[126] Z. Li, C. Wang, and R. Xu. Computation offloading to save energy on handheld devices: a partition scheme. In *Proc. ACM Int. Conf. Compilers, Architecture, and Synthesis for Embedded Systems*, pages 238–246, Nov. 2001.

[127] J. Lorch. A complete picture of the energy consumption of a portable computer. Master's thesis, Computer Science Dept., University of California at Berkeley, 1995.

[128] J. Lorch and A. J. Smith. Operating system modifications for task-based speed and voltage scheduling. In *Proc. USENIX/ACM Int. Conf. Mobile Systems, Applications, & Services*, pages 215–229, May 2003.

[129] J. R. Lorch and A. J. Smith. Scheduling techniques for reducing processor energy use in MacOS. *Wireless Network*, 3(5):311–324, Oct. 1997.

[130] J. R. Lorch and A. J. Smith. Improving dynamic voltage scaling algorithms with PACE. In *Proc. ACM Int. Conf. Measurement & Modeling of Computer Systems*, pages 50–61, June 2001.

[131] J. R. Lorch and A. J. Smith. Using user interface event information in dynamic voltage scaling algorithms. In *Proc. IEEE/ACM Int. Symp. Modeling, Analysis & Simulation of Computer and Telecommunications Systems*, pages 46–55, Oct. 2003.

[132] Y.-H. Lu, L. Benini, and G. De Micheli. Dynamic frequency scaling with buffer insertion for mixed workloads. *IEEE Trans. Computer-Aided Design of IC & Systems*, 21(11):1284–1305, Nov. 2002.

[133] Y.-H. Lu, L. Benini, and G. De Micheli. Power-aware operating systems for interactive systems. *IEEE Trans. VLSI Systems*, 10(2):119–134, Apr. 2002.

[134] Z. Lu, J. Hein, M. Humphrey, M. Stan, J. Lach, and K. Skadron. Control-theoretic dynamic frequency and voltage scaling for multimedia workloads. In *Proc. ACM Int. Conf. Compilers, Architecture, and Synthesis for Embedded Systems*, pages 156–163, Oct. 2002.

[135] J. Luo and N. K. Jha. Power-conscious joint scheduling of periodic task graphs and aperiodic tasks in distributed real-time embedded systems. In *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, pages 357–364, Nov. 2000.

[136] J. Luo and N. K. Jha. Battery-aware static scheduling for distributed real-time embedded systems. In *Proc. ACM/IEEE Design Automation Conf.*, pages 444–449, June 2001.

[137] K. Lyons, T. Starner, D. Plaisted, J. Fusia, A. Lyons, A. Drew, and E. W. Looney. Twiddler typing: One-handed chording text entry for mobile phones. In *Proc. Conf. Human Factors in Computing Systems*, pages 671–678, Apr. 2004.

[138] I. S. MacKenzie and W. Buxton. Extending Fitts' law to two-dimensional tasks. In *Proc. Conf. Human Factors in Computing Systems*, pages 219–226, May 1992.

[139] I. S. MacKenzie and R. W. Soukoreff. Text entry for mobile computing: Models and methods, theory and practice. *Human-Computer Interaction*, 17(2-3):147–198, 2002.

[140] I. S. MacKenzie and C. Ware. Lag as a determinant of human performance in interactive systems. In *Proc. Conf. Human Factors in Computing Systems*, pages 488–493, 1993.

[141] T. L. Martin and D. P. Siewiorek. Nonideal battery and main memory effects on CPU speed-setting for low power. *IEEE Trans. VLSI Systems*, 9(1):29–34, Feb. 2001.

[142] MiBench. *http://www.eecs.umich.edu/mibench/*.

[143] Microsoft SPOT. *http://www.spot-watch.com/*.

[144] Microsoft Transcriber. *http://www.microsoft.com/windowsmobile/downloads/transcriber.mspx/*.

[145] Microsoft Voice Command. *http://www.microsoft.com/windowsmobile/downloads/voicecommand/*.

[146] A. Miyoshi, C. Lefurgy, E. V. Hensbergen, R. Rajamony, and R. Rajkumar. Critical power slope: Understanding the runtime effects of frequency scaling. In *Proc. Int. Conf. Supercomputing*, pages 35–44, June 2002.

[147] Mobile phones move deeper into PDA turf. *http://news.com.com/2102-1041_3-5065145.html*. Aug. 2003.

[148] S. Mohapatra, R. Cornea, N. Dutt, A. Nicolau, and N. Venkatasubramanian. Integrated power management for video streaming to mobile handheld devices. In *Proc. Ann. ACM Int. Conf. Multimedia*, pages 582–591, Nov. 2003.

[149] D. Mosse, B. Aydin, B. Childers, and R. Melhem. Compiler-assisted dynamic power-aware scheduling for real-time applications. In *Proc. Wkshp. Compilers & Operating Systems for Low Power*, Oct. 2000.

[150] Motorola Offspring Concept Design, 2003. *http://www.phonescoop.com/articles/moto_wearables/*.

[151] S. S. Muchnick. *Advanced Compiler Design and Implementation*. Morgan Kaufmann Publishers Inc., San Francisco, CA, 1997.

[152] A. Muttreja, S. Ravi, A. Raghunathan, and N. K. Jha. Automated performance/energy macro-modeling of embedded software. In *Proc. ACM/IEEE Design Automation Conf.*, pages 99–102, June 2004.

[153] B. A. Myers. User interface software tools. *ACM Trans. Computer-Human Interaction*, 2(1):64–103, Mar. 1995.

[154] B. A. Myers and M. B. Rosson. Survey on user interface programming. In *Proc. Conf. Human Factors in Computing Systems*, pages 195–202, May 1992.

[155] D. Narayanan and M. Satyanarayanan. Predictive resource management for wearable computing. In *Proc. USENIX/ACM Int. Conf. Mobile Systems, Applications, & Services*, pages 113–128, May 2003.

[156] Nemoptic Advanced LCD Technologies. *http://www.nemoptic.com*.

[157] R. Neugebauer and D. McAuley. Energy is just another resource: Energy accounting and energy pricing in the Nemesis OS. In *Proc. Wkshp. Hot Topics in Operating Systems*, pages 59–64, May 2001.

[158] B. D. Noble, M. Satyanarayanan, D. Narayanan, J. E. Tilton, J. Flinn, and K. R. Walker. Agile application-aware adaptations for mobility. In *Proc. ACM Symp. Operating Systems Principles*, pages 276–287, Oct. 1997.

[159] T. Okuma, T. Ishihara, and H. Yasuura. Real-time task scheduling for a variable voltage processor. In *Proc. Int. Symp. System Synthesis*, pages 24–29, Nov. 1999.

[160] N. Omoigui, L. He, A. Gupta, J. Grudin, and E. Sanocki. Time-compression: Systems concerns, usage, and benefits. In *Proc. Conf. Human Factors in Computing Systems*, pages 136–143, May 1999.

[161] Opie. *http://opie.handhelds.org*.

[162] M. Othman and S. Hailes. Power conservation strategy for mobile computers using load sharing. *SIGMOBILE Mobile Computing and Communication Review*, 2(1):44–51, Jan. 1998.

[163] Palm OS development. *http://www.palmsource.com/developers/*.

[164] A. E. Papathanasiou and M. L. Scott. Energy efficient prefetching and caching. In *Proc. USENIX Ann. Tech. Conf.*, pages 255–268, June-July 2004.

[165] T. Pering, T. Burd, and R. Brodersen. The simulation and evaluation of dynamic voltage scaling algorithms. In *Proc. Int. Symp. Low Power Electronics & Design*, pages 76–81, Aug. 1998.

[166] T. Pering, T. Burd, and R. Brodersen. Voltage scheduling in the IpARM microprocessor system. In *Proc. Int. Symp. Low Power Electronics & Design*, pages 96–101, July 2000.

[167] Philips' display driver solutions for mobile applications. *http://www.semiconductors.philips.com/acrobat/literature/ 9397/75009652.pdf*.

[168] P. Pillai and K. G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *Proc. ACM Symp. Operating Systems Principles*, pages 89–102, Oct. 2001.

[169] J. Pouwelse, K. Langendoen, and H. Sips. Dynamic voltage scaling on a low-power microprocessor. In *Proc. ACM Ann. Int. Conf. Mobile Computing & Networking*, pages 251–259, July 2001.

[170] J. Pouwelse, K. Langendoen, and H. Sips. Energy priority scheduling for variable voltage processors. In *Proc. Int. Symp. Low Power Electronics & Design*, pages 28–33, Aug. 2001.

[171] J. Pouwelse, K. Langendoen, and H. J. Sips. Application-directed voltage scaling. *IEEE Trans. VLSI Systems*, 11(5):812–826, Oct. 2003.

[172] Q. Qiu, Q. Wu, and M. Pedram. Stochastic modeling of a power-managed system: Construction and optimization. *IEEE Trans. Computer-Aided Design of IC & Systems*, 20(10):1200–1217, Oct. 2001.

[173] Qt. *http://www.trolltech.com/*.

[174] Qt/Embedded. *http://www.trolltech.com/download/qt/embedded.html*.

[175] Qtopia. *http://www.trolltech.com/products/qtopia/*.

[176] G. Qu. What is the limit of energy saving by dynamic voltage scaling? In *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, pages 560–563, Nov. 2001.

[177] L. Rabiner and B.-H. Juang. *Fundamentals of Speech Recognition*. Prentice-Hall, Inc., 1993.

[178] V. Raghunathan, T. Pering, R. Want, A. Nguyen, and P. Jensen. Experience with a low power wireless mobile computing platform. In *Proc. Int. Symp. Low Power Electronics & Design*, pages 363–368, Aug. 2004.

[179] T. V. Raman. *Auditory User Interfaces: Toward the Speaking Computer*. Kluwer Academic Publishers, Boston, MA, 1997.

[180] D. Ramanathan, S. Irani, and R. Gupta. Latency effects of system level power management algorithms. In *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, pages 350–356, Nov. 2000.

[181] A. Rudenko, P. Reiher, G. J. Popek, and G. H. Kuenning. Saving portable computer battery power through remote process execution. *SIGMOBILE Mobile Computing and Communication Review*, 2(1):19–26, Jan. 1998.

[182] H. Saputra, M. Kandemir, N. Vijaykrishnan, M. J. Irwin, J. S. Hu, C.-H. Hsu, and U. Kremer. Energy-conscious compilation based on voltage scaling. In *Proc. Joint Conf. Languages, Compilers & Tools for Embedded Systems*, pages 2–11, June 2002.

[183] N. Sawhney and C. Schmandt. Speaking and listening on the run: Design for wearable audio computing. In *Proc. IEEE Int. Symp. Wearable Computers*, page 108, Oct. 1998.

[184] B. Schneiderman. Response time and display rate in human performance with computers. *ACM Comput. Surv.*, 16(3):265–285, Sept. 1984.

[185] R. Schneiderman. Bluetooth's slow dawn. *IEEE Spectrum*, pages 61–65, Nov. 2000.

[186] A. Sears and Y. Zha. Data entry for mobile devices using soft keyboards: Understanding the effects of keyboard size and user tasks. *Int. J. Human-Computer Interaction*, 16(2):163–184, 2003.

[187] P. Shenoy and P. Radkov. Proxy-assisted power-friendly streaming to mobile devices. In *SPIE/ACM Conf. Multimedia Computing and Networking*, Jan. 2003.

[188] E. Shih, P. Bahl, and M. J. Sinclair. Wake on wireless: An event driven energy saving strategy for battery operated devices. In *Proc. ACM Ann. Int. Conf. Mobile Computing & Networking*, pages 160–171, Sept. 2002.

[189] Y. Shin and K. Choi. Power conscious fixed priority scheduling for hard real-time systems. In *Proc. ACM/IEEE Design Automation Conf.*, pages 134–139, June 1999.

[190] B. Shneiderman. Touch screens now offer compelling uses. *IEEE Software*, 8(2):93–94, 107, Mar. 1991.

[191] B. Shneiderman. *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Addison Wesley Longman, Reading, MA, 3rd edition, 1998.

[192] B. Shneiderman. *Leonardo's Laptop: Human Needs and the New Computing Technologies*. The MIT Press, Oct. 2002.

[193] D. P. Siewiorek. New frontiers of application design. *Communication of ACM*, 45(12):79–82, Dec. 2002.

[194] A. Sinha and A. Chandrakasan. JouleTrack - A web based tool for software energy profiling. In *Proc. ACM/IEEE Design Automation Conf.*, pages 220–225, June 2001.

[195] Source code for benchmarks. *http://www.ee.princeton.edu/∼cad/benchmarks.html*.

[196] M. B. Srivastava, A. P. Chandrakasan, and R. W. Brodersen. Predictive system shutdown and other architectural techniques for energy efficient programmable computation. *IEEE Trans. VLSI Systems*, 4(1):42–55, Jan. 1996.

[197] T. Starner. Keyboards redux: Fast mobile text entry. *IEEE Pervasive Computing*, (3):97–101, July-Sept. 2004.

[198] M. Stemm and R. H. Katz. Measuring and reducing energy consumption of network interfaces in hand-held devices. *IEICE Trans. Communications*, E80-B(8):1125–1131, 1997.

[199] B. Suhm, B. Myers, and A. Waibel. Multimodal error correction for speech user interfaces. *ACM Trans. Computer-Human Interaction*, 8(1):60–98, 2001.

[200] SUIF 2 Compiler System. *http://suif.stanford.edu/suif/suif2/*.

[201] T. K. Tan, A. Raghunathan, and N. K. Jha. EMSIM: An energy simulation framework for an embedded operating system. In *Proc. Int. Symp. Circuits & Systems*, pages 464–467, May 2002.

[202] T. K. Tan, A. Raghunathan, G. Lakshminarayana, and N. K. Jha. High-level software energy macro-modeling. In *Proc. ACM/IEEE Design Automation Conf.*, pages 605–610, June 2001.

[203] The Embedded Linux GUI/ Windowing Quick Reference Guide. *http://www.linuxdevices.com/articles/AT9202043619.html*.

[204] The Familiar project. *http://familiar.handhelds.org*.

[205] The Skiff cluster. *http://www.handhelds.org/projects/skiffcluster.html*.

[206] Thumbscript. *http://www.thumbscript.com*.

[207] V. Tiwari, S. Malik, and A. Wolfe. Power analysis of embedded software: A first step towards software power minimization. *IEEE Trans. VLSI Systems*, 2(4):437–445, Dec. 1994.

[208] Transmeta LongRun Technology. *http://www.transmeta.com/crusoe/longrun.html*. Jan. 2000.

[209] Twiddler. *http://www.handykey.com*.

[210] T. Šimunić, L. Benini, P. Glynn, and G. De Micheli. Event-driven power management. *IEEE Trans. Computer-Aided Design of IC & Systems*, 20(7):840–857, July 2001.

[211] T. Šimunić, G. De Micheli, L. Benini, and M. Hans. Source code optimization and profiling of energy consumption in embedded systems. In *Proc. Int. Symp. System Synthesis*, pages 193–199, Sept. 2000.

[212] A. Vahdat, A. Lebeck, and C. S. Ellis. Every joule is precious: the case for revisiting operating system design for energy efficiency. In *Proc. ACM SIGOPS European Wkshp.*, pages 31–36, Sept. 2000.

[213] K. S. Vallerio, L. Zhong, and N. K. Jha. Energy-efficient graphical user interface design. *IEEE Trans. Mobile Computing: to appear*, 2005.

[214] M. A. Viredaz and D. A. Wallach. Power evaluation of a handheld computer. *IEEE Micro*, 23(1):66–74, Jan./Feb. 2003.

[215] R. Want, T. Pering, G. Dianneels, M. Kumar, M. Sundar, and J. Light. The personal server: Changing the way we think about ubiquitous computing. In *Proc. Int. Conf. Ubiquitous Computing*, pages 194 – 209, Sept.-Oct. 2002.

[216] A. Wasserman. Extending state transition diagrams for the specification of human-computer interaction. *IEEE Trans. Software Engineering*, 11(8):699–713, Aug. 1985.

[217] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for reduced CPU energy. In *Proc. Symp. Operating Systems Design & Implementation*, pages 13–23, Nov. 1994.

[218] X Window System. *http://www.x.org*.

[219] F. Xie, M. Martonosi, and S. Malik. Compile-time dynamic voltage scaling settings: Opportunities and limits. In *Proc. ACM SIGPLAN Conf. Programming Languages Design & Implementation*, pages 49–62, June 2003.

[220] L. Yan, L. Zhong, and N. K. Jha. User-perceived latency driven voltage scaling for interactive applications. In *Proc. ACM/IEEE Design Automation Conf.*, pages 624 – 627, June 2005.

[221] W. Ye, N. Vijaykrishna, M. Kandemir, and M. Irwin. The design and use of SimplePower: A cycle-accurate energy estimation tool. In *Proc. ACM/IEEE Design Automation Conf.*, pages 340 – 345, June 2000.

[222] G. K. Yeap. *Practical Low Power Digital VLSI Design*. Kluwer Academic Publishers, Norwell, MA, 1998.

[223] W. Yuan and K. Nahrstedt. Energy-efficient soft real-time CPU scheduling for mobile multimedia systems. In *Proc. ACM Symp. Operating Systems Principles*, pages 149–163, Oct. 2003.

[224] W. Yuan and K. Nahrstedt. Practical voltage scaling for mobile multimedia devices. In *Proc. Ann. ACM Int. Conf. Multimedia*, pages 924–931, Oct. 2004.

[225] W. Yuan, K. Nahrstedt, S. Adve, D. Jones, and R. Kravets. GRACE: Cross-layer adaptation for multimedia quality and battery energy. *IEEE Trans. Mobile Computing: to appear*, 2005.

[226] ZBD Displays Ltd. *http://www.zbddisplays.com*.

[227] J. Zedlewski, S. Sobti, N. Garg, F. Zheng, A. Krishnamurthy, and R. Wang. Modeling hard-disk power consumption. In *Proc. USENIX Conf. File & Storage Technologies*, pages 217–230, Mar.-Apr. 2003.

[228] H. Zeng, C. S. Ellis, A. R. Lebeck, and A. Vahdat. ECOSystem: Managing energy as a first class operating system resource. In *Proc. Int. Conf. Architectural Support for Programming Languages & Operating Systems*, pages 123–132, Oct. 2002.

[229] Zenithal: Bistable Displays. *http://www.zbddisplays.com*.

[230] F. Zheng, N. Garg, S. Sobti, C. Zhang, R. E. Joseph, A. Krishnamurthy, and R. Y. Wang. Considering the energy consumption of mobile storage alternatives. In *Proc. IEEE/ACM Int. Symp. Modeling, Analysis & Simulation of Computer and Telecommunications Systems*, pages 36–45, Oct. 2003.

[231] Y. Zheng, Z. Liu, Z. Zhang, M. Sinclair, J. Droppo, L. Deng, A. A. Acero, and X. Huang. Air-and bone-conductive integrated microphones for robust speech detection and enhancement. In *Proc. IEEE Wkshp. Automatic Speech Recognition & Understanding*, pages 249–254, Nov.-Dec. 2003.

[232] L. Zhong and N. K. Jha. Graphical user interface energy characterization for handheld computers. In *Proc. ACM Int. Conf. Compilers, Architecture, and Synthesis for Embedded Systems*, pages 232–242, Nov. 2003.

[233] L. Zhong and N. K. Jha. Dynamic power optimization for interactive systems. In *Proc. Int. Conf. VLSI Design*, pages 1041–1047, Jan. 2004.

[234] L. Zhong and N. K. Jha. Energy efficiency of handheld computer interfaces: Limits, characterization, and practice. In *Proc. USENIX/ACM Int. Conf. Mobile Systems, Applications, & Services*, pages 247–260, June 2005.