CARNEGIE MELLON UNIVERSITY

Balancing Batteries, Power, and Performance: System Issues in CPU Speed-Setting for Mobile Computing

A DISSERTATION
SUBMITTED TO THE GRADUATE SCHOOL
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree of

DOCTOR OF PHILOSOPHY
in
ELECTRICAL AND COMPUTER ENGINEERING

by

Thomas L. Martin

Advisor: Professor Daniel P. Siewiorek

Pittsburgh, Pennsylvania
August, 1999

Typographical corrections, January, 2001

Abstract

This thesis studies the problem of balancing power and performance in mobile computers, specifically, trading off power for performance by CPU speed-setting. The traditional approach to power-performance trade-offs assumes that batteries and memory bandwidth are ideal and focuses on lowering the energy per operation. This research, however, shows that non-ideal battery and performance behavior must be considered to properly balance power and performance, and that computations per battery life is a better metric for power-performance trade-offs than energy per operation.

The thesis begins with a description of non-ideal battery properties that can affect power-performance trade-offs and then presents models for those properties. The models delineate regions where batteries can be treated ideally and where their non-ideal behavior must be considered. Furthermore, the models show that peak power rather than average power determines the available battery capacity. Thus, the first major result is that decreasing a mobile computer's active power will increase the battery life more than decreasing its idle power, even if both reduce the average power by the same amount.

The thesis then shows that the memory system also has an impact on CPU speed-setting. Because of limits in memory bandwidth, code performance will not scale with CPU speed when there are a considerable number of accesses to main memory. The second major result is to show that, because of non-ideal memory performance and non-ideal battery capacity, the results of some experiments are nearly a factor of four less for a real system than what would be expected using the ideal assumptions. For those experiments, the computations per discharge is expected to increase by 230%, but instead the measured results show a 37% decrease.

Consequently, a system-level approach to CPU speed-setting should account for the non-idealities of both the memory and the battery. The final major result is an outline of a realistic method for CPU speed-setting, one that accounts for non-ideal memory and battery behavior by using performance-monitoring registers and battery "gas gauge" integrated circuits.

Acknowledgments

My advisor and committee were the people most directly involved with the completion of my dissertation. Dan Siewiorek patiently shaped this work as it developed through a series of false starts and dead ends. I benefited greatly from his ability to approach problems from many different directions. Dan's sense of the importance of personal relations over research concerns will be something for me to live up to as I set out on my own. As for my committee, Don Thomas gave me a great deal of latitude when I was his teaching assistant, and gave me a fine example of how to blend zest and humor in the classroom with high standards for his students. Phil Koopman sent me a steady stream of related work, prodded me with questions, and answered my questions, both technical and professional. Mootaz Elnozahy always had an open door when he was on campus, and his questions while I was writing helped me remember that there was a big picture and what it looked like.

A dissertation is the culmination of years of work, not just at the institution where the degree is granted, and not just by the person writing it. My graduate studies would not have been possible without the able teaching of Dennis Cornelius, Gary Weiss, Ranga Vemuri, and Phil Wilsey. Mr. C. drove me over most of Ohio for science fairs and lectures, and taught me that where there's smoke there's science; Gary whipped up a pile of recipes in his calculus kitchen that whetted my mental appetite; Ranga designed a new VLSI curriculum and taught most of it to me; and Wilsey first introduced me to computer architecture and gave me sound advice over the years.

My work could not have been completed without a substantial amount of funding, hardware, and software from outside sources, for which I am grateful. The National Science Foundation funded my studies for three years with a fellowship, and funded the Engineering Design Research Center which was a home to me and the wearable computing project of which I was a member. Compaq's

Western Research Lab donated two Itsy computers to me, without which I could not have completed this work, and the members of the Itsy team provided technical support, especially Bill Hamburgen, Marc Viredaz, and Debby Wallach. Marc Doyle of the Department of Chemical Engineering at U.C. Berkeley provided the Li-ion battery model described in Chapter 3. Mark Weiser and Brent Welch of Xerox PARC gave me their software and traces for low power CPU scheduling.

A number of people at Carnegie Mellon provided advice and support over the years, both technical and personal. Gennady Neplotnik gave me a detailed etymology of the Russian saying in Chapter 3. Randy Casciola helped me with hardware and equipment, Rhonda Moyer solved my procurement problems, and Laura Forsyth found Dan for me when I needed him; all became my good friends over the years. Matt Weiss and Josh Silverman were always ready to roll up their sleeves and go to the archives with me for research, a necessary complement to my practical work in the lab. A string of office mates answered my questions and put up with my (sugar and writing induced) ramblings: Jim Beck, Forrest Chamberlain, and Jason Lee in the early days, and later Grace McNally, LeMonté Green, John Dorsey, and Jolin Warren. The late Mrs. Lydia Gugliotta gave me the pen that I used while filling notebooks over the years and writing the rough drafts of the dissertation.

My parents and siblings began my education by teaching me to talk, much to their later chagrin. They also bought me a telescope, took me to Cape Kennedy, and looked the other way when I dismantled clocks and radios, teaching me lessons that I could not have learned in a classroom. They smiled when I told them repeatedly for over a year that I needed only two more months to finish, smiling as only people who love me could.

Finally, Karen O'Kane became the love of my life when I came to graduate school. I would not have finished without her. We were a two-dissertation household, not always an easy situation, but she helped me through it with laughter, fortitude, and love.

Thank you, all of you.
TLM
March 13, 1999

Contents

Chapter 1

Introduction

"Energy is eternal delight."
William Blake

The ongoing miniaturization of electronic components has fueled the growth of the mobile computing industry over the last decade [9]. Hand-held and wearable computers are now available with computing performance comparable to desktop systems of only a few years ago. A major design constraint of the designers of mobile systems is the size and weight of batteries [65]. The size and weight limitations are mainly due to two factors. The first is the relatively slow growth of the specific energy and energy density of the battery technologies [9], which are within a factor of 2-3 of their theoretical limits [15][52]. The second is the increasing power consumption of the hardware [9]. Mobile system designers have typically approached the problem by attempting to reduce the power consumption for a given level of performance or to increase the performance for a given level of power consumption, as exemplified by [9]. The result has been that a typical high-end notebook computer had a battery life of less than 3 hours in both 1991 and 1999: 25 out of 27 notebooks in a 1991 review had battery lives of less than 3 hours [64], as did 25 out of 51 notebooks tested in a 1999 review [57]. (45 out of 51 had lives of less than 4 hours.) Users of portable systems find that there is not enough energy available for them to have eternal delight.

The goal of this work is to incorporate models of power source behavior into existing system models in order to better understand the trade-offs between power and performance. The motivating observation is that the capacity of a battery, the amount of energy delivered before the battery must be recharged, tends to decrease as the power of the battery load increases. Currently, designers of mobile computers implicitly assume that the battery capacity is constant for all loads, i.e. that the energy delivered by the battery does not depend on the rate at which the energy is delivered, by using energy per operation to evaluate power-performance trade-offs. If battery capacity were taken into account, these trade-offs would sometimes be made differently. This thesis introduces the idea that, once the system response time is adequate, the number of computations performed in a discharge cycle is the major concern of users. The computations per discharge is the metric by which power-performance trade-offs should be judged, as it captures important aspects of both battery capacity and system performance.

The computations per discharge is essentially the battery's capacity divided by the energy per operation for a given computation. When evaluating a proposed modification *x* from a system perspective, the three major factors in the computations per discharge are system power as a function of the modification, *SystemPower(x)*; battery capacity as a function of system power, *BatteryCapacity(SystemPower(x))*; and application performance as a function of the modification, *Performance(x)*. The relationship between these factors and the computations per discharge is given by

$$Computations\ \ per\ \ Discharge(x) = \frac{BatteryCapacity(SystemPower(x))}{SystemPower(x)} \times Performance(x)$$

where the battery capacity is given in Watt-hours per discharge, the power in Watts, and the performance in computations per hour. Then the units of *BatteryCapacity(SystemPower(x))/SystemPower(x)* is hours per discharge, which when multiplied by *Performance(x)* gives units of computations per discharge.

This dissertation will focus mainly on the battery capacity function, describing the regions of operation where it is ideal and where it is non-ideal, providing analytical models and practical rules for mobile system designers. This dissertation will also discuss typical assumptions about system power and performance, and when these assumptions may be invalid. The three factors will be used to consider the problem of CPU speed-setting, first showing that previous approaches have been simplistic, and then outlining the elements of a more realistic approach.

CPU speed-setting has been chosen as the example for the dissertation because assuming ideal battery behavior results in a much different solution than if battery behavior is non-ideal. All of the previous work on the problem has stated that it is “useless” to set the CPU speed at less than its maximum frequency if the CPU voltage is not also reduced. This dissertation will show simple cases where setting the CPU speed is not useless, even with a fixed voltage, because of non-ideal battery and performance behavior. Finally, the dissertation proposes a realistic policy for CPU speed-setting, one that takes into account both non-ideal battery and performance behavior.

1.1 Scope of this research

This research spans several areas of interest to designers of mobile systems. Figure 1.1 shows the hierarchy of those areas. The areas above the shaded line are concerned with *power consumers*. A common feature of the previous work in these areas is the emphasis on lowering average power. The areas below the shaded line are related to the *power sources.*

The top level of the consumers hierarchy is low power software, which includes reducing power consumption by changing algorithms, reducing performance needs, and re-compiling to use lower power instructions. The bottom level of the consumer hierarchy, low power hardware, includes providing new mechanisms for power savings to the higher levels and implementing novel circuit structures or devices. Power management, the explicit

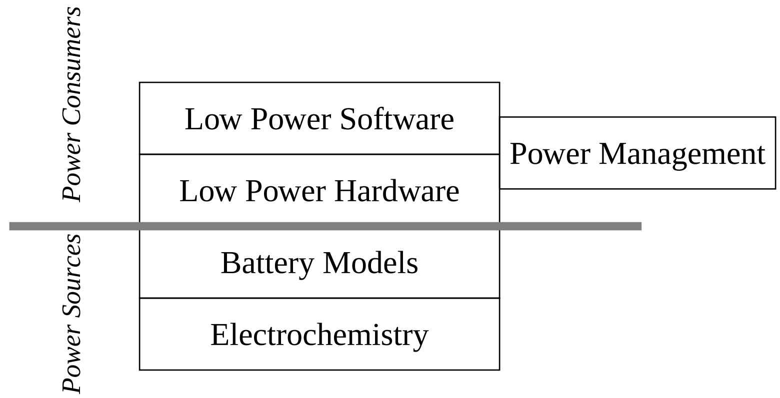


Figure 1.1 Hierarchy of Areas of Related Work

scheduling of device accesses and shutdowns to save power, may be implemented in hardware, software, or some combination of the two.

The upper level of the power sources hierarchy, battery modeling, deals mainly with abstracting from the behavior of batteries to predict their discharge times. Most battery models have been created to aid battery designers in optimizing cell parameters. The lower level of the source hierarchy is electrochemistry. Here new battery families are created, increasing the energy per mass and energy per volume available for mobile systems. The research at this level also includes safety and environmental issues, cyclability (the number of charge-discharge cycles a battery can withstand), and manufacturability.

This dissertation bridges the boundary between sources and consumers, incorporating knowledge of the source behavior into the behavior of the consumers where appropriate. This differs from the most of the previous work in the area, which, with the exception of battery monitoring hardware [4][24] and some recent work in setting operating voltage levels for CMOS integrated circuits [10][58], failed to make a connection between consumers and sources.

1.2 Organization

The dissertation takes a "bottom-up" approach, treating each of the factors of computations per discharge separately before dealing with them together. The organization of the remainder of the dissertation is as follows:

Chapter 2 covers the related work in low power hardware and software and in battery modeling.

Chapter 3 discusses the power sources, including battery behavior and models for simulating that behavior. The models are then used to show that peak power rather than average power determines battery capacity.

Chapter 4 explores the performance and power of a system as a function of its CPU clock frequency using an analytical expression called the *work ratio* and presents the results of continuous discharge experiments using a variety of systems.

Chapter 5 presents a generalized form of the work ratio, discusses useful system features to support CPU speed-setting, and describes an operating system policy for setting the CPU speed dynamically.

Chapter 6 summarizes the contributions of the thesis and discusses avenues for future work in the area. The latter includes investigating the relationship between bounds on computing power and power sources, creating a framework for profiling application power usage, and incorporating models of battery behavior in other areas, such as autonomous mobile robotics.

1.3 Research Contributions

The major contribution of this research is to show that non-ideal battery and system behavior must be considered when making power-performance trade-offs in mobile computing. The specific contributions to the areas of low power and mobile computer system design include the following:

- Regions of ideal/non-ideal battery behavior are delineated, allowing system designers to understand when measuring only average power is adequate, and when dynamic power must also be considered.
- Typical bounds on the approximate effect of battery capacity loss are given.
- Continuous discharge behavior is shown to be a better estimate of the computations completed during an intermittent discharge than average power.
- Reducing idle power is found to have less of an effect than reducing active power when the active power lies in the non-ideal range of battery behavior.
- It is the only system level power research to use battery discharge experiments to verify predictions.
- The memory hierarchy is shown to be an important consideration when setting the CPU speed.
- A realistic policy for dynamically setting the CPU speed is described.

Chapter 2

Related Work

"In making the handle of an axe,
By cutting wood with an axe
The model is indeed near at hand."
--Lu Ji, as translated by Gary Snyder

The three areas most closely related to the topic of this thesis are low power software, low power hardware, and battery modeling. Low power software and hardware lie in the upper portion of the hierarchy shown in Chapter 1 (power consumers), while battery modeling lies in the lower portion (power sources). The last section of the chapter covers work that spans the gap between the consumers and sources.

2.1 Low power software

The work in the area of low power software includes power-savings and awareness in applications, compilation, and operating systems. The consensus is that the potential for power savings in software is greater than the potential for savings in hardware, but that the software savings are more difficult to achieve [40]. Hence, what is evident in the following sections is that more work has been done in saving power at the lower levels of the software, i.e. at the operating systems and compilation levels.

The potential for algorithmic changes to save power is considered to be large. The choice of algorithm constrains the hardware and power savings achieved in hardware in two ways: First, if one algorithm takes longer to execute than another, then the system will consume more energy because it is active longer. Second, if an algorithm blindly uses resources, those resources cannot be put into low power idle modes. A trivial example of this is a routine which polls a device rather than using interrupts. The related work at the algorithmic level thus looks to reduce performance and resource requirements. At this time, researchers are still trying to gain an understanding of how application software affects power consumption. To that end, Flinn and Satyanarayanan describe a tool for profiling the energy usage of applications for mobile computing and correlating dynamic power traces to procedure call traces [23], Ong and Yan analyze the memory usage of basic searching and sorting algorithms and the usage's effect on power consumption [54], and Wuytack et al. discuss memory transformations to lower communications costs [73].

Most of the work in low power software has been in the operating system (OS), specifically, utilizing power management features of the underlying hardware. The OS is in a better position to judge whether a device should be put into a low power mode than the device itself is, because it has a view of the overall state of the system. The OS is also in a better position to judge than an application because it can balance the needs of several applications. Furthermore, if the OS makes the power management decisions then the applications do not need to be modified. The Advanced Power Management specification allows the OS to manage low power hardware, especially in Microsoft Windows [36]. Lorch and Smith consider transition strategies for a number of different subsystems, especially within the constraints of the design philosophy of MacOS [45]. Paleologo et al. consider the overhead requirements of making a transition from one power management state to another and introduce a stochastic model for evaluating power management policies [55]. Hardware aspects of power management will be considered in Section 2.2.

Most of the work in power issues of the operating system has been concerned with power management; very little has considered power-performance trade-offs. The major power-performance trade-off under OS control to be treated previously is CPU speed-setting, a topic central to this dissertation.

The previous work in CPU speed-setting [59][72][74] makes the following assumptions:

- Performance is proportional to clock frequency f.
- Power is proportional to fCV^2, where C is capacitance and V is voltage.

Given these two assumption, the previous work shows that if the voltage V is held constant while the clock frequency is changed, then the energy per operation is constant for all frequencies. If there are other subsystems besides the CPU, then the energy per operation decreases as the frequency increases and the CPU should be run as fast as possible. Therefore, according to the previous work, reducing the CPU frequency will not save energy if the voltage is held constant while the frequency is changed. For speed-setting to be useful, the CPU must decrease the voltage as well as the frequency [72]. If both are changed by a factor s, then the energy per operation will change by a factor of s^2. Thus running as slowly as possible will minimize the energy per operation. The previous work describes policies for setting the speed based upon these assumptions [31][72]. Each of the policies looks at windows of time in the range of 10 ms to 100 ms, and in each window of time tries to run as slowly as possible, i.e. to reduce the idle time to zero. If the speed is set too slow to complete all the work for the current window, then the speed is increased to complete that work in the next window of time. In this way, latency can only increase by the window size. The major differences between the policies is in the prediction of the work for the next window of time. The common feature of all the policies is that the idle time is the only variable they consider when determining the CPU speed, which neglects several practical aspects of the problem, as this dissertation will show.

Weiser et al. introduce several policies for setting CPU speed [72]. The major problem for them is predicting the amount of work to be completed in the next window. Any work left over from the current window because of running too slowly has to be completed in the next window, thus maintaining the response time as seen by the user. Running too slowly is found to increase the energy consumption because of the faster speed required in the next window to complete any leftover cycles. Weiser et al. uses trace-driven simulation to judge the quality of the predictions for several policies, showing savings of up to 70% for CPU energy consumption. The set of traces was collected from engineering workstations.

Several other groups have since offered improvements to the policies of Weiser et al. Govil et al. use the same traces and assumptions as Weiser et al. to present several new policies for making more accurate predictions about activity [31]. Improvements on the results of Weiser et al. are on the order of 10% of CPU energy consumption. Yao et al. prove the minimum schedule is to run as slowly as possible given that the energy per operation varies as s^2 [74]. They do not consider the general case where there are lower order terms in the function for energy per operation. Pering and Brodersen examine speed-setting policies in from a real-time perspective [59]. Their scheduling approach is not significantly different than Weiser's. Their main contribution is to look at a set of software that they consider to be more typical of the mobile environment than Weiser's traces were.

The final area of low power software is compilation for reduced power consumption. Tiwari et al. examine instruction-level power consumption for both the Intel x86 architecture and a Fujitsu RISC architecture [68][69]. The power consumption for each instruction is measured, as are the inter-instruction and data contributions. The compiler is modified to take into account the power consumption as well as the timing cost of each instruction. The major energy savings come from reducing the time to complete a computation, not from using lower power instructions. The peak power for their test cases typically increases, but the cycle count for the programs is reduced by a much larger amount, so that the overall energy consumption is reduced.

2.2 Low power hardware

While a great deal of work has been done in low power software, by far the largest amount of low power research has been in the hardware area.

The power management software described in the previous section requires hardware support. Examples of hardware power management are detailed in [8], [11], [29], and [47]. Bhattacharya gives an overview of power management techniques to educate designers using Intel's 386SL processor [8]. Child draws parallels between low power techniques used by hand calculator manufacturers in the 1970's and those used by the notebook computer makers of the 1990's [11]. Glass gives an excellent subsystem-by-subsystem overview of power management techniques [29]. Martin provides a case study in reducing power consumption via a systematic evaluation of the subsystems [47].

The low-power hardware area most closely related to this thesis is the design and implementation of variable-voltage CPU's. Kuroda et al. describe their implementation of a variable-voltage RISC processor [42]. They implemented the voltage modification by replicating the critical path of the CPU and adding several gate delays to it. When the speed and voltage change, this replicated path is checked for failure. When lowering the voltage, the replicated critical path fails before the real critical path because of the additional gate delays. When the replicated circuit fails, the voltage is raised a step to make sure both critical paths work. When raising the voltage and frequency, as soon as the replicated circuit begins working the voltage is no longer increased. The interesting point of the implementation from the viewpoint of this thesis is that the plot of the CPU's power consumption versus frequency, shown in Figure 2.1, has a large y-intercept, about 10% of the total power at the CPU's highest speed. As will be shown later in this thesis, having a large y-intercept significantly limits the total savings a speed-setting policy can achieve and determines which policies will be effective for the system.

Another area of low power hardware is reversible and adiabatic computing. Theories of reversible computing are tangential to this thesis, but are noted here because of the possible implications that fundamental limits on computing may have for power-performance trade-offs. Bennett and Landauer argue that there is no fundamental lower limit on the amount of energy needed to create information [5][6]. Destroying information does

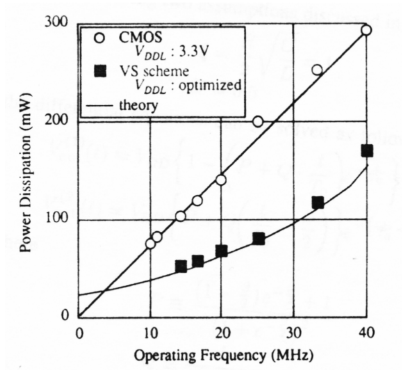


Figure 2.1 Power versus frequency for variable-voltage CPU (after [42])

require energy, however. One of the noteworthy points of Bennett's arguments is that in order to create information without consuming energy, the computation must proceed arbitrarily slowly. He cites the need for a "driving force" to make the computation proceed in the correct direction. The harder the computation is driven (e.g. the faster it is executed) the more energy it must consume. Thus there may be a fundamental trade-off between performance and power. Several other works look at the fundamental limits on computation [22][38][39][50] and implementations of reversible computing circuits and CPU's using adiabatic techniques [3][41][75]. None of the work in fundamental limits of computing have considered limits in the human-computer interface such as the minimum power for a sound to be heard or a display to be seen, an oversight if those limits are to be applied to mobile computing.

2.3 Battery models

There are two main classes of battery models, constant load and intermittent load. All of the intermittent load models can handle constant loads, and so only the earliest constant load model will be presented. The models report either the battery's charge capacity or its energy capacity. (The charge capacity is the amount of charge the battery delivers in a discharge cycle, while the energy capacity is the amount of energy delivered [43]. Both are more fully described in Chapter 3.)

2.3.1 Constant load model

Peukert determined the nonlinearity of batteries for constant loads in the late 1800's. Peukert's formula for the charge capacity Q of a battery is $Q = k/I^{\alpha}$, where k is a constant determined by the materials and physical design of the battery and I is the load current. For an ideal battery, $\alpha = 0$, i.e. the capacity is constant, but for real batteries, α ranges between 0.2 and 0.7 for most loads, and like k is determined by the chemical family and physical design of the battery [43]. Given this relationship between charge capacity and load current, the relation between discharge time and load power is $T = k'/P^{(1+\alpha)}$, where T is the discharge time, k' is a constant, and P is the load power. Peukert's formula may not hold for intermittent loads. The conditions under which Peukert's formula may be used for intermittent loads is described in Chapter 3.

2.3.2 Variable load models

Four variable-load models were considered:

- Kinetic Battery Model (KiBaM) [46]
- SPICE model [33]
- Battery Energy Storage Test (BEST) model [35]
- Doyle's low-level electrochemical model of lithium-ion (Li-ion) cells [15][16]

The majority of the variable load models have been created for either lead-acid storage battery facilities or for low-level cell design. With the exception of the SPICE model, none had been created for use by mobile system designers. And the SPICE model was not created for Li-ion cells.

Doyle's model will be the only one used to study intermittent operation in this dissertation. However, the others will be described briefly as they could form the groundwork of a "black box" model in the future. For determining whether or not battery properties can affect power-performance trade-offs, Doyle's model inspired more confidence, having been correlated to a variety of Li-ion cell families from several battery manufacturers and having been used by one of them to explore cell design parameters [18] and another to verify a method for determining capacity versus discharge rate [19]. In comparison, KiBaM, the SPICE model, and the BEST model seemed ad hoc and possibly untrustworthy, especially since they had not been intended to be used with Li-ion cells.

The only drawback of Doyle's model is the large number of parameters needed, the majority of which are trade secrets and cannot be measured by the user. For the purposes of this thesis, KiBaM, the SPICE model, and the BEST model have elements that may be the basis of a phenomenological model more suitable to the needs of mobile system designers than Doyle's low-level model. Consequently, a description of each is included here.

2.3.3 Kinetic Battery Model (KiBaM)

The Kinetic Battery Model, KiBaM, was intended for use with large lead-acid storage batteries [46]. It models the battery as two wells of charge, as shown in Figure 2.2. The available-charge well supplies electrons directly to the load; the bound-charge well supplies electrons only to the available-charge well. The rate of charge flow between the two wells is set by k' and the difference in the heights of the two wells, h_1 and h_2. The state of charge of the battery is h_1, i.e., when h_1 is unity the battery is fully charged and when it is zero the battery is fully discharged. The internal resistance of the battery is represented by R_0.

KiBaM needs a number of additions to be useful for the types of batteries used in mobile computing. For example, it used a simple linear relation between the state of charge of the battery and the battery's open-circuit voltage, which is sufficient for lead-acid batteries with their flat discharge profiles. To adequately model many families of Li-ion batteries, with their sloped discharge profiles, KiBaM needs a more complex relation between the state of charge and open-circuit voltage.

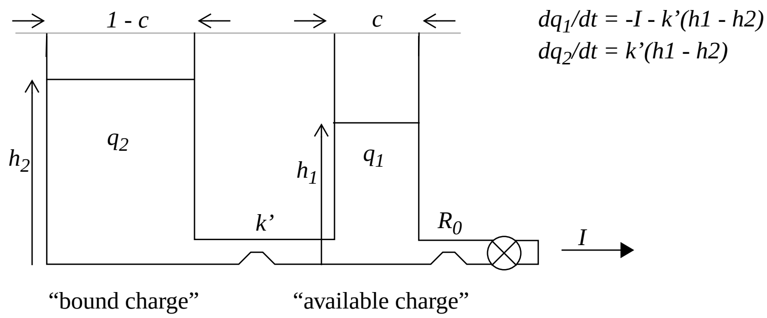


Figure 2.2 The Kinetic Battery Model, KiBaM, models a battery as two wells of charge, available and bound. After [46].

But KiBaM is useful for an intuitive sense of why the recovery effect can occur. (Recovery is explained in Section 3.2.) Suppose a load is attached and I is large. The available-charge well will quickly be reduced, and the difference in h_1 and h_2 will be large. Now the load I is removed. Charge flows from the bound-charge well to the available-charge well until h_1 and h_2 are equal. The battery's open-circuit voltage increases, and more charge is available to the load than would have been if it had been connected continuously until h_1 went to zero.

2.3.4 SPICE model

The SPICE model simulates alkaline, nickel-cadmium (NiCd), and lead-acid batteries [33]. The elements of the model are shown in Figure 2.3. The output voltage of the battery is determined by a table lookup of the battery's state of charge. The state of charge is determined by the charge remaining on the capacitor *C_Capacity* and the current discharge rate. The elements R_2 and C_1 model the delay of the battery in responding to a changing load. Because the SPICE model did not include Li-ion parameters and because some of its elements (notably R_2 and C_1) seemed ad hoc and difficult to determine, it was deemed untrustworthy for exploring intermittent operation. But like KiBaM, it could serve as the basis for a model in the future with some modification.

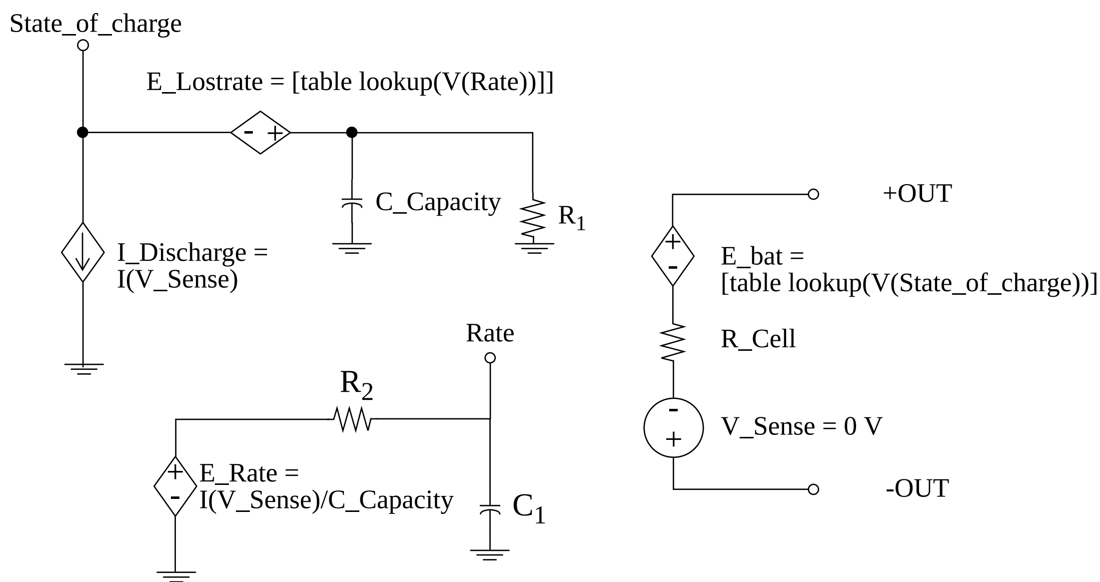


Figure 2.3 Elements of the SPICE model

2.3.5 Battery Energy Storage Test (BEST) model

The Battery Energy Storage Test (BEST) model, like KiBaM, is intended for use with large lead-acid storage batteries. BEST is based upon the curves of the voltage versus current at different levels of depth of discharge shown in Figure 2.4. BEST assumes these curves are parallel, which may be true of lead-acid batteries with their relatively flat volt-

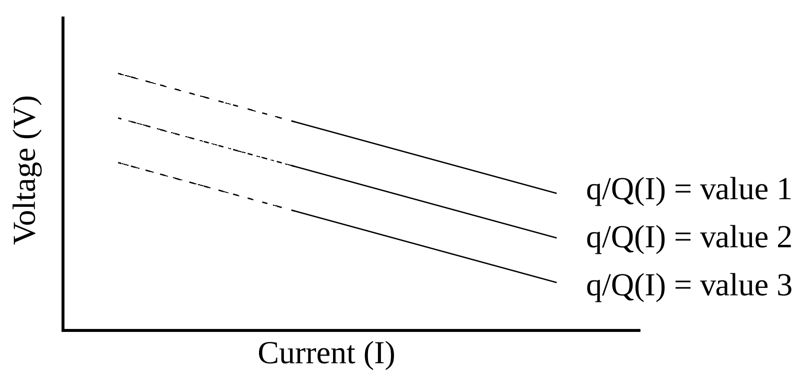


Figure 2.4 Voltage versus current at different depths of discharge, after [35]

age discharge curves and when the cutoff voltage is a function of the discharge rate. But in general, these curves are not parallel [44] and the BEST model will fail. But one interesting features of the BEST model is that it represents a normalized depth of discharge as a function of both the amount of charge delivered and the rate of the discharge,

$$X(q, I) = q/Q_0 + qI/Q_0I_0$$

where X is the normalized depth of discharge, q is the amount of charge delivered, I is the rate of the discharge, Q_0 is the limiting charge capacity at very low rates, and I_0 determines the loss of capacity with increasing load. Hence the state of charge depends not only on how much charge has been delivered but also on how quickly it is being delivered. Another interesting feature is that for variable loads the dynamic response of the voltage increases as the depth of discharge approaches unity, approximating the apparent increase in internal resistance near the end of discharge displayed by many types of batteries. Both of these features could perhaps be combined with the table-lookup representation of voltage from the SPICE model to create a model which could handle batteries with flat voltage profiles and those with sloped voltage profiles.

2.3.6 Doyle's model

Doyle's model is a first-principles electrochemical model, unlike the ones above. The model uses concentrated solution theory [52], and solves a set of six equations describing the current, mass transport, reactant concentrations, and potentials using finite difference methods [16]. (For a brief review of battery properties modeled by Doyle, see Appendix C.) Similar models have been created for NiCd [13] and alkaline batteries [61]. Doyle's model is intended to aid battery designers, not battery users.

2.4 Crossing the boundary between battery and hardware/software

This chapter has described the related work in low power software, low power hardware, and battery modeling. Before closing, it should be noted that none of the work described above crosses the boundary between the battery and the software/hardware, which is the central topic of this thesis. The earliest efforts to cross the boundary were probably by the manufacturers of battery "gas-gauge" integrated circuits [4]. These limited themselves to providing information to notebook computers about the charge remaining in the batteries, and only lately have added features to predict the time remaining at a given rate of discharge. The earliest known work to emphasize the importance at looking at computations per discharge rather than energy per operation and to make a tie between loss of battery capacity and power-performance trade-offs was the published form of the proposal for this thesis [48]. Pedram's group has begun to look at loss of battery capacity and VLSI design [10][58]. There are three major differences between their work and the work described in this thesis. First, they make assumptions similar to those of Weiser et al. about the system power being proportional to fCV^2 in order to find the optimal voltage for a CMOS IC. Second, they use linear and quadratic approximations of battery capacity that are apparently not based on experimental data. Finally, they do not consider possible intermittent battery phenomena. In the area of mobile computer networking, Zorzi and Rao look at the energy consumption of wireless communication protocols and point out that the loss of battery capacity could have an impact on the protocols [76]. However, they propose a Markov model that does not include the battery effects, but leave it for future work to include states to account for the loss of battery capacity. Thus they have recognized that the battery properties must be considered but have not considered them in their current approach.

Chapter 3

Battery behavior and modeling

In principle, yes; in practice, no.
Russian saying

This chapter describes the behavior of batteries and simulation results generated by Doyle's first-principles battery model. The first part of the chapter deals with the characteristics of an ideal battery and how these characteristics are used in sizing batteries and estimating discharge times. Then the chapter discusses typical non-ideal characteristics and gives a description of the regions of operation where they occur. Assumptions about batteries to be used throughout the dissertation are then given. The second part of the chapter covers results from Doyle's battery model, showing likely areas for exploiting battery behavior in mobile computing.

3.1 Ideal battery properties and discharge time estimates

The two most important properties of batteries from the viewpoint of someone using them are voltage and capacity. An ideal battery has a constant voltage throughout a discharge, which drops instantaneously to zero when the battery is fully discharged, and has constant capacity no matter what the rate of the load, as shown in Figure 3.1.

For sizing batteries, the battery voltage should be in the allowable range of the power supply of the device in question. The battery voltage is considered to be the rated voltage of the battery, e.g. 1.2V per cell for nickel-cadmium (NiCd) and nickel-metal hydride

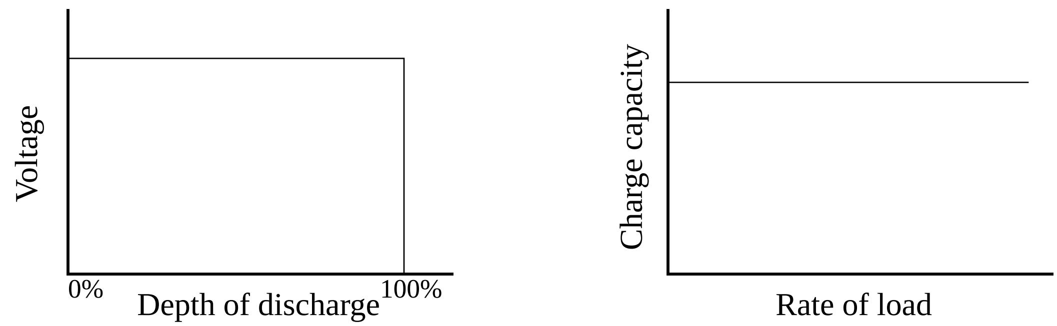


Figure 3.1 Characteristics of an ideal battery: Constant voltage and constant capacity

(NiMH) batteries and 3.6V per cell for most lithium-ion (Li-ion) batteries. The charge capacity of the battery is typically given in terms of Amp-hours or milliAmp-hours and is called the battery's "C" rating. The C rating is used in the battery industry to normalize the load current to the battery's capacity [33][43]. For example, a load current of 1C for a battery with a C rating of 500 mA-hours is 500 mA, while a load current of 1C for a battery with a C rating of 1000 mA-hours would be 1000 mA. A load current of 0.1C is 50 mA for the former and 100 mA for the latter. The advantage of C ratings is that it allows battery manufacturers to present one graph of discharge curves for batteries of similar construction but different capacities. The C rating is specified as the capacity for a given time of discharge.

Two methods are used to estimate discharge time, depending on the type of load. If the load is a constant current load, then the discharge time T is estimated to be the charge capacity C divided by the load current I, or $T = C/I$. If the load is a constant power load, then the discharge time T is estimated to be the battery's rated voltage V multiplied by the charge capacity C, divided by the average power P of the load, or $T = (C \times V)/P$. The rated voltage multiplied by the charge capacity is the battery's nominal energy capacity, typically given in Watt-hours (1 Wh = 3600 J). As Section 3.5 will show, these methods will overestimate the battery life if the load has a large peak value.

3.2 Non-ideal battery properties

While ideally a battery has constant voltage and capacity, in practice both vary widely. Figure 3.2a shows the battery voltage as a function of discharge time for two different loads. The load on the battery for discharge curve 1 is smaller than load for discharge curve 2. Because of resistance and other losses, the voltage throughout the discharge is lower for curve 2 than curve 1. The voltage for each load also drops over the course of the discharge due to changes in the active materials and reactant concentrations [43].

The capacity also varies with the value of the load. The two major ways in which it varies are loss of capacity with increasing load, and an effect called recovery where an intermittent load may have a larger capacity than a continuous load. Figure 3.2b shows the loss of capacity with increasing load current for a typical NiCd battery [43]. The capacity decreases by about 40% over a range of discharge times of 10 hours (0.1C discharge rate) to 0.1 hours (10C discharge rate). As stated previously, the C rating is specified as the capacity for a given time of discharge. The capacity in Figure 3.2b was measured at the 2 hour rate, since 100% capacity occurs at 0.5C. If the capacity had been measured at the 10 hour rate, 100% would have occurred at 0.1C.

The second non-ideal capacity property, recovery, is shown in Figure 3.2c [43]. A reduction of the load for periods of time results in an increase in battery capacity. The voltage rises while the load is reduced, and the overall time of discharge increases. This phenomenon occurs because, during the time when load is reduced, reactants in the battery diffuse to the reaction location, allowing more of them to be used during the life of the battery. The degree to which the battery recovers depends on the discharge rate and the length of time the load is reduced, as well as the details of the battery construction.

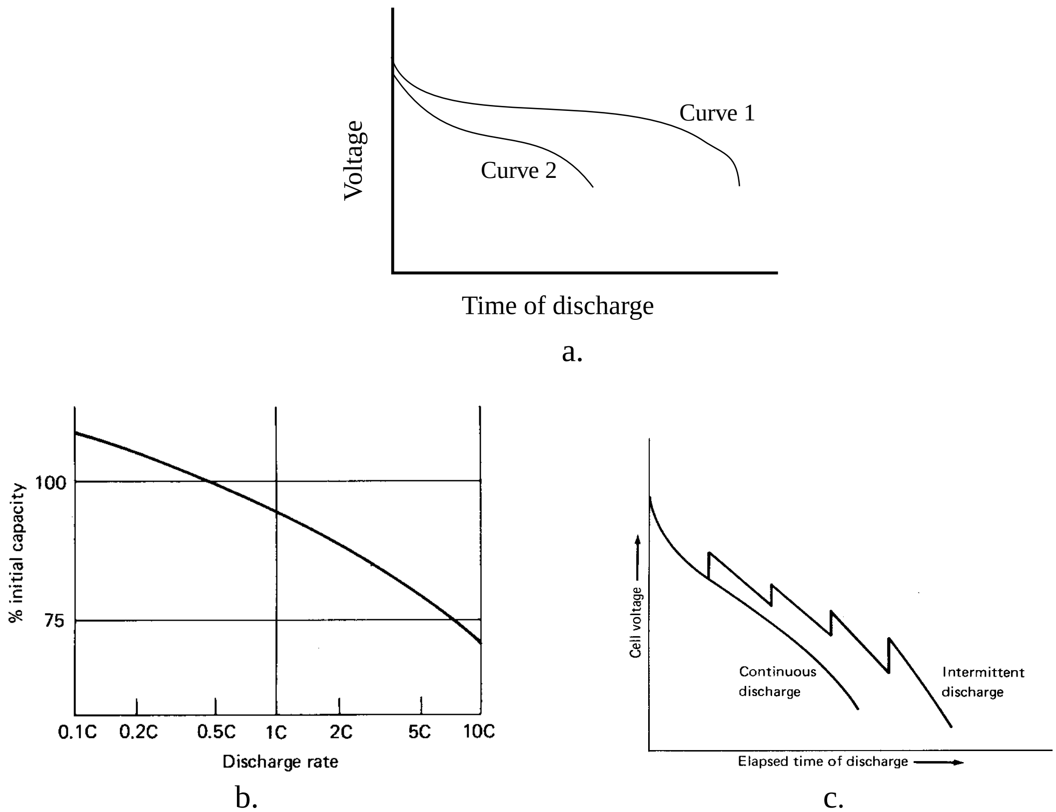


Figure 3.2 Non-ideal battery properties: (a) voltage change, (b) loss of capacity, and (c) recovery (after [43])

It is widely known that the battery voltage varies during discharge. For example, power supplies are usually rated over a range of input voltages. When a power supply is used with a battery, it is necessary to ensure that the range of the supply's input voltage includes the range of the battery voltage during discharge. Since the voltage variation is widely known, this dissertation will not focus on it. The non-ideal capacity properties, on the other hand, are not widely known, and so will be one of the main subjects of the remainder of this work. Given how a battery's discharge time is estimated using ideal values of voltage and capacity, the loss of capacity can lead to an overestimate of the discharge time for large loads. While using a chart such as Figure 3.2b allows accounting for the loss of capacity for loads that are constant and continuously on, in general loads are variable and

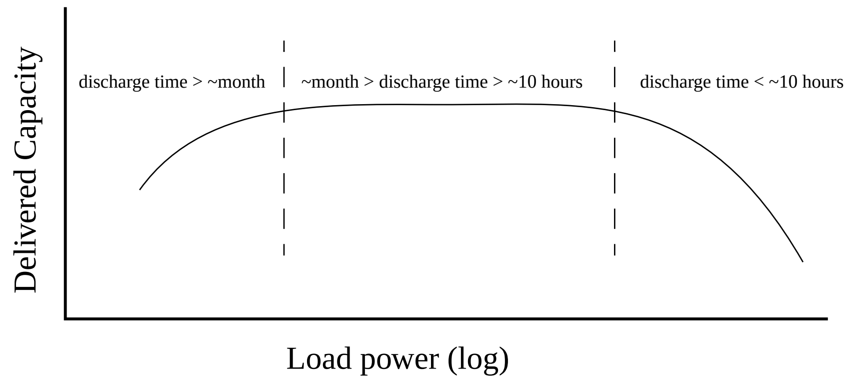


Figure 3.3 Battery capacity versus load power over wide range of loads. (Note that values given are typical of Li-ion cells but will vary depending upon the details of the battery construction.)

intermittent. If recovery occurs, then the duration of the off times of the load must be considered in addition to the duration of its on times and its value while on. Models that account for both capacity loss and recovery are needed to properly simulate the loads encountered in mobile computing. But before examining results from Doyle's model for non-ideal battery behavior, it is beneficial to describe some general regions of operation.

3.3 Regions of operation

Figure 3.3 shows the battery capacity versus the load over a broad range of loads. At very low rates, where the discharge time is measured in weeks (months or years for some types of batteries), the capacity is reduced because of self-discharge mechanisms. Over the broad middle range of load power, the capacity is constant. At high rates, the capacity begins to decrease because of the loss of capacity shown in Figure 3.2b. Table 3.1 lists the four regions of operation, each requiring a different model of battery behavior, covering these ranges of load power. The regions shown in Table 3.1 cover most of the cases of interest for mobile computing. Note that the values given are generalizations, to give the reader an understanding of the order of magnitude of the characteristics. The exact values will depend on the battery's active materials and the details its construction

Table 3.1: Regions of operation and corresponding battery models

Type of load	Model
I. Average and peak load << self-discharge rate	Constant life
II. Self-discharge rate ≅ average/peak load < 0.1C	Constant energy
III. Constant or variable load with f > 1 Hz, ave. & peak > 0.1C	Constant load
IV. Variable load, frequency < 1 Hz	Variable load

[17][26][43][53]. Figure 3.4 shows typical capacity versus power curves for a number of battery families, showing the wide variation due to the choice of active materials. The capacity versus power will also vary for two cells with the same active materials but different construction, as illustrated by Figure 3.5 [43]. One cell has been optimized for low-rate discharges, while the other has been optimized for high-rate discharges. Thus the exact

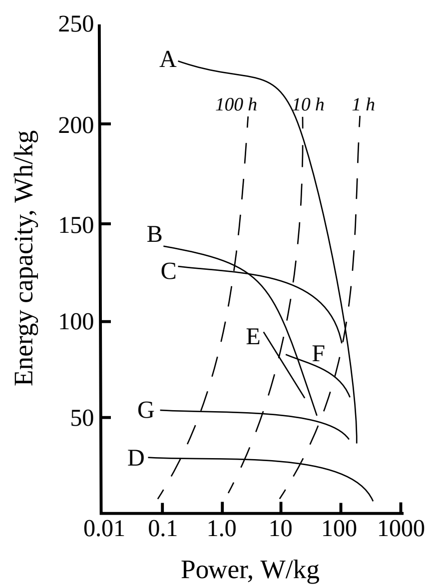


Figure 3.4 Capacity versus power for several battery systems. A: Li/MnO_2 2/3A cell; B: Zn/alkaline/MnO_2 AA cell; C: Li-MnO_2 AA cell; D: Ni-Cd AA cell; E: Zn/alkaline/MnO_2 AA cell; F: Li-ion AA cell; G: NiMH AA cell. A and B are primary cells, C-G are secondary cells. Dashed lines show time of discharge for reference. (After [43].)

values of the regions in Table 3.1 will depend on the particular battery in question. The general values given in Table 3.1 are those of a “typical” Li-ion cell, the battery of choice for most of the mobile computing market at present, as will be explained in Section 3.4.

In region I of Table 3.1, the load value is much less than the self-discharge rate of the battery. All batteries have self-discharge mechanisms by which the battery loses charge while no load is connected to it. The self-discharge rate of a battery depends mainly on the battery’s chemistry, and partially on the details of its construction. For nickel cadmium batteries, the self-discharge rate is approximately 20-30% of capacity per month. In contrast, self-discharge rates for lithium watch batteries are much lower, allowing them to be stored for years with only a 10-20% loss of capacity. When the load is much less than this self-discharge rate, then the self discharge rate dominates and the battery can be viewed as having a constant life. This region is included for completeness but is not encountered often in practice. When the load is much less than the self-discharge rate, it is usually best to find a battery with a much lower self-discharge rate if possible.

In region II, the load is of the same order of magnitude or larger than the self-discharge rate, but less than approximately 0.1C. Then the battery can be viewed as being ideal, with a constant energy capacity. Most power-performance trade-offs are made using this

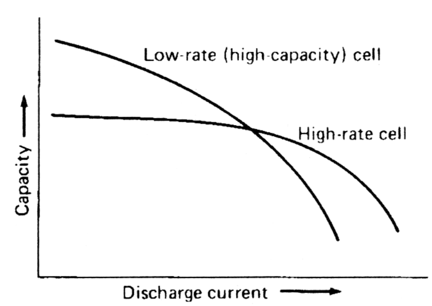


Figure 3.5 Capacity versus power depends on cell design parameters, even if active materials are the same. (After [43].)

assumption, even in systems where the load is much greater than 0.1C. Since the battery has constant energy, then the discharge time is the energy capacity divided by the average power of the load.

Region III will be one of the concerns of this dissertation. If the load value is greater than 0.1C, and the load is either constant or variable with a frequency greater than approximately 1 Hz, then the load can be viewed as being constant and the battery capacity as being dependent on the load. (Because of the time constants involved in the battery reactions, load frequencies greater than approximately 1 Hz are essentially filtered out and can be replaced by their average value [33].) One model for this region is Peukert's model, discussed in Section 2.3.1.

This dissertation will also focus on region IV, where the load is greater than 0.1C and slowly varying, at a frequency of less than 1 Hz. A variable-load model must be used for these types of loads. Several models for variable load operation exist. One of the goals of this work is to find one appropriate for the types of loads and batteries seen in mobile systems. Several possible models were presented in Section 2.3.

The details of the underlying mechanisms of these regions vary with battery family, but in general the mechanisms are related to the rate at which reactions take place in the battery and the rate at which reactants move to or from the reaction boundaries. Self-discharge rates are governed by "side" reactions, unintended reactions between materials in the cell. The loss of capacity for discharge rates above 0.1C is due partly to internal resistance and partly to the difference between the rate at which the current producing reaction takes place and the rate at which reactants diffuse to the reaction site. When the concentrations at the reaction site drop below critical values, the cell voltage decreases abruptly and the cell cannot be used further without damaging it. For variable loads, the reaction boundary appears electrically to be a capacitance set up by the charges of the ions, which acts as a low pass filter. Consequently, the average value of a high frequency intermittent load can be used to predict the discharge time. More detailed information on the electrochemical mechanisms involved can be found in Appendix C.

3.4 Battery-related Assumptions to Be Used Throughout the Dissertation

Throughout this work there are several assumptions about the type of load and the type of battery. This section describes the types of loads and types of batteries that are typical of mobile computing, and then specifies the load and battery types employed for the remainder of this dissertation.

3.4.1 Type of load

There are three major types of loads presented to batteries: Constant resistance, constant current, and constant power [43]. ("Constant" here refers to how the load behaves when it is of a given value, not to its being a given value continuously.) An example of a constant resistance load is a flashlight. A constant current load would be one that uses a linear voltage regulator. Finally, a constant power load is one that uses a switching power supply whose efficiency is nearly constant over the range of battery voltage during discharge. The current, voltage, and power during each of these types of discharge is shown in Figure 3.6, with each type of discharge having equal power at the beginning of discharge. Because the voltage decreases as the battery discharges, each type of discharge has different current and power profiles during the discharge. The charge capacity delivered with each type of load is roughly the same, so the constant resistance load will have the longest discharge time as its average current is lowest, and the constant power load will have the shortest discharge time as its average current is highest. It is necessary to specify which type of load was used to calculate the capacity rating of a battery. Typically, battery manufacturers determine the charge capacity of their cells using a constant current load. The energy

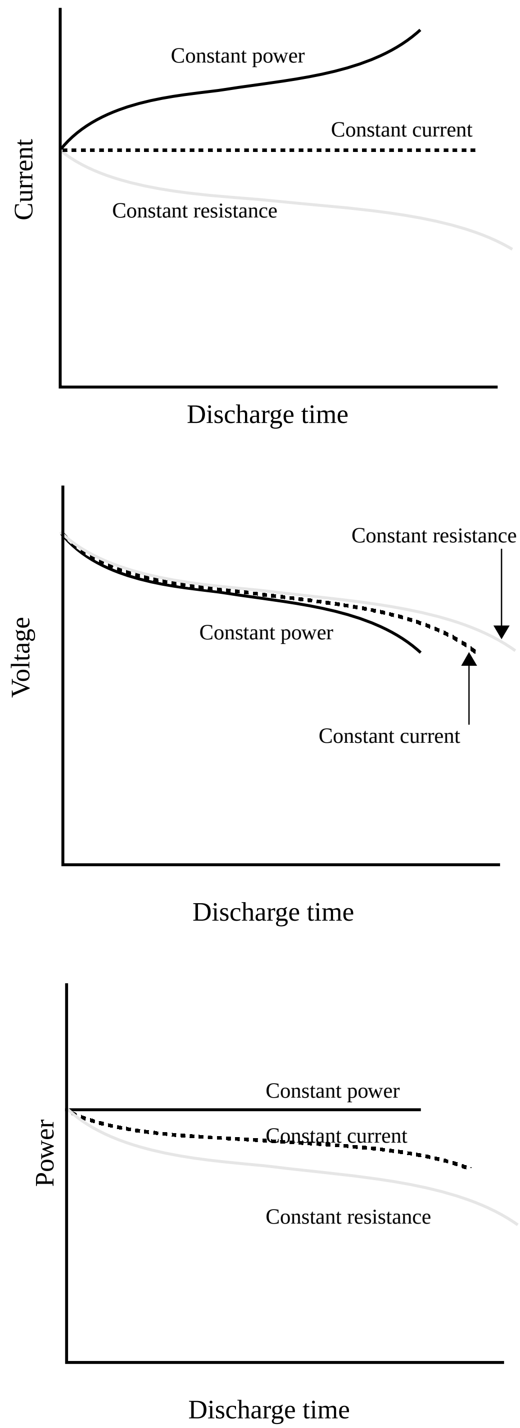


Figure 3.6 Properties of different types of loads with the same power at the beginning of discharge, after [43]

capacity is then estimated by multiplying the nominal voltage by the charge capacity. More importantly for this dissertation, most mobile systems use switching power supplies, and so are constant power loads. Therefore, when the equations and simulations in the remainder of this work assume a constant load, it will be a constant power load.

3.4.2 Type of battery

This dissertation will concentrate on rechargeable Li-ions, the battery of choice for present-day mobile computers. Two other battery families, NiCd and NiMH, are also very common in the mobile market, but Li-ion is becoming more common due to its superior specific energy and energy density. Concerns about safety and difficulty of charging have been Li-ion's major drawbacks. These were overcome as the technology matured. Sony introduced the first Li-ion phone cell in 1990, and several other companies have begun to offer Li-ion products as well. Barring a breakthrough, Li-ion will remain the power source of medium to high-end mobile products for the next decade. Only very price sensitive products and those requiring high discharge rates (discharge time less than 1 hour) will use NiCd or NiMH. Many of the non-ideal properties discussed in this chapter are true for all three families. When a statement is made that depends on a property that is true only of Li-ion, it will be noted.

3.5 Results with Doyle's variable load model

The typical load of a mobile computer system is not constant, but variable. A model is needed, then, to estimate the discharge time with variable loads. A variable-load model will ideally possess the following characteristics:

- Relatively accurate capacity information (i.e. if several loads are simulated, then the model should correctly predict the relative difference in discharge times, even if the actual differences are inaccurate.)
- Applicable to a variety of battery types

- Intuitive parameters and behavior
- Ease of correlation to actual cells

Of these four criteria, the first is the most important for this dissertation. The last three will become more important when battery models are more widely used in mobile system design. Consequently, of the four variable-load models discussed in Chapter 2, this dissertation will focus on Doyle's model. Doyle's model inspired the most confidence due to its having been created solely for Li-ion cells and due to its use in industry. The others had not been created for use with Li-ion cells and hence results with them would have required lengthy correlation with actual cells before their predictions could have been trusted.

Doyle's model was used to study the effect of intermittent discharges on the capacity and demonstrates that peak power predicts battery capacity better than average power. Figure 3.7 shows the model results for battery capacity versus average power for continuous discharges over a range of loads, and for intermittent discharges at duty cycles of 25, 50, and 75% and three values of peak power. The intermittent discharges were square waves with an off power of 0 W/kg. The two major features of the results are that the capacity decreases as the load power increases for continuous loads, and that there is a range where the peak power of an intermittent load rather than the average power is a stronger indicator of the battery's capacity. For example, the 300 W/kg continuous load results in a battery capacity of 90 Wh/kg (point A in the figure) and the 75 W/kg continuous load results in a battery capacity of 140 Wh/kg (point B), while the intermittent load with a peak power of 300 W/kg and duty cycle of 25% (i.e. an average power of 75 W/kg, point C) results in a capacity of approximately 100 Wh/kg. Thus using the average power of this intermittent load would over-estimate the battery capacity by about 40% (i.e., point B's 140 Wh/kg would be expected), while using the peak power would under-estimate it by only about 10% (i.e., point A's 90 Wh/kg would be expected). To put these results in more common terms, the 75 W/kg continuous load B would have a battery life of about 1.9 hours, while the intermittent load C, with the same average power, 75 W/kg, would have a battery life

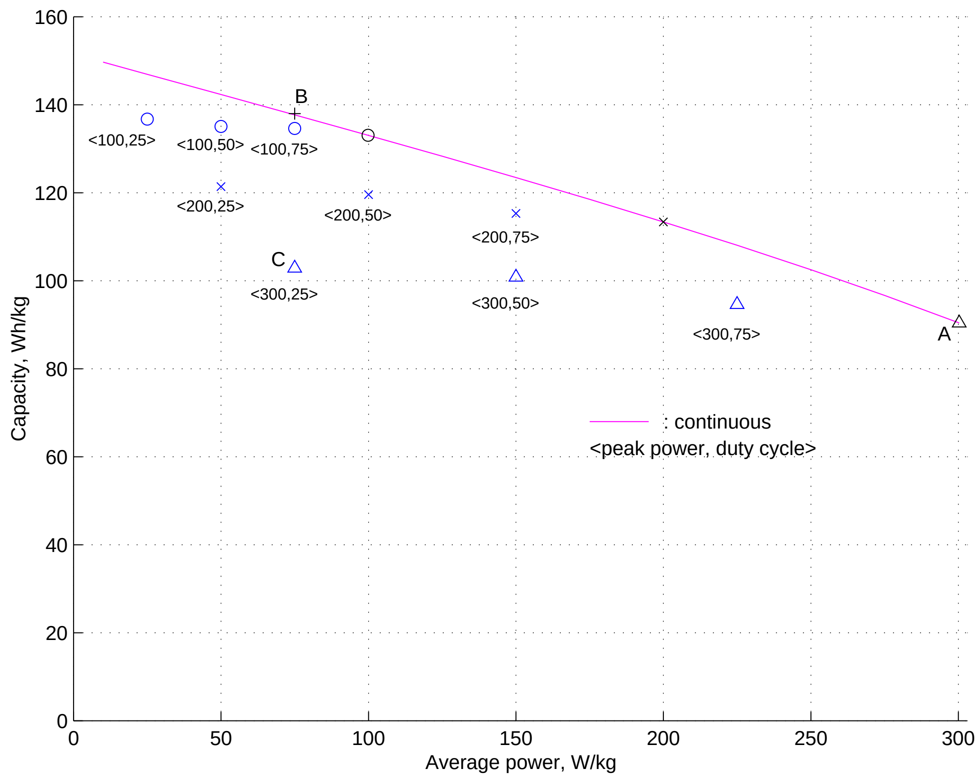


Figure 3.7 Doyle's Li-ion model results for capacity versus average power, showing difference between continuous and intermittent loads of same average value

of about 1.3 hours. Only when the peak power is below about 50 W/kg (about a 3 hour discharge when continuously on) would the peak and average power give about the same estimate of battery life.

While the amount of capacity lost with increasing load will vary from battery family to battery family, the general trend will be true across families. To verify that the trends were not simply a property of the model, intermittent discharge experiments were performed using a commercially available Li-ion battery, the Sony LIP-2000, with a chemistry similar to the one of the model [25]. The same general trends were visible, as shown in Figure 3.8. While it is difficult to make one-to-one comparisons between the two figures because the model accounted for only the active materials of the battery (hence the reason for the model results being given in W/kg and Wh/kg, rather than simply W and Wh), the loss of capacity for continuous loads and capacity limited by peak power can be observed in both figures. For example, the capacity delivered by a continuous load of about 10.5 W is about

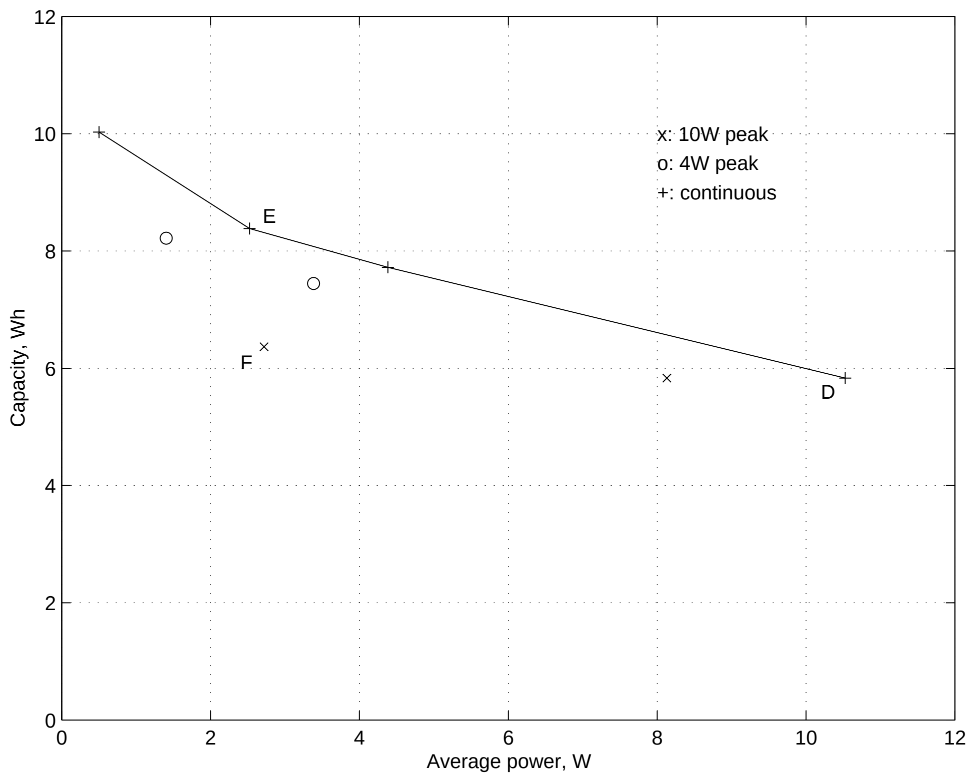


Figure 3.8 Measured Li-ion battery results for capacity versus average power with Sony LIP-2000 batteries, showing difference between continuous and intermittent loads of same average value

6 Wh (point D) and the capacity delivered by a continuous load of about 2.5 W is about 8 Wh (point E), but the capacity of an intermittent load with a peak of 10.5W and an average value of approximately 2.5 W is about 6 Wh (point F). So if one had estimated the battery

life for the intermittent load of point F using only its average power, the estimate would have be wrong by about 25%, even if the loss of capacity at that average power were accounted for.

To further test the effect of the recovery phenomena, the intermittent load simulations described above were run with waveforms of different periods. Figure 3.9 shows the simulation results for periods of 10 seconds and 1000 seconds. The largest difference between the two is about 10%, much less than the loss of capacity. If the difference were due to recovery, one would expect the load with the 1000s period to have a greater capacity, due to its longer off-time. The results show that the opposite is true: The load with the shorter period has greater capacity. The reason for this is that the 1000 second period has a longer

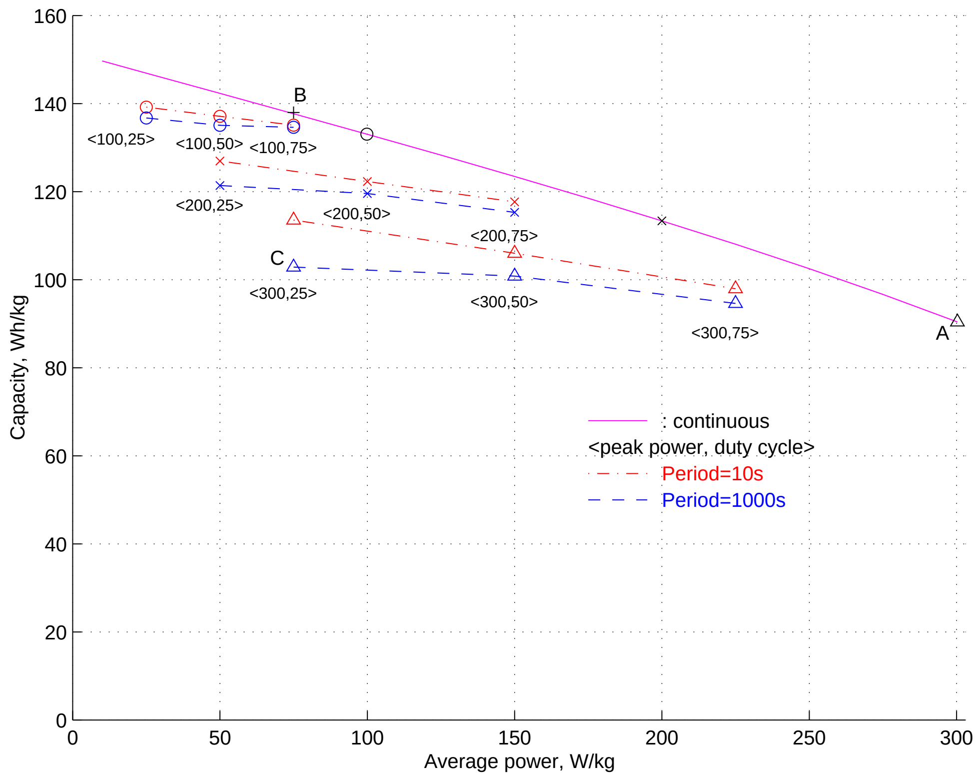


Figure 3.9 Doyle's Li-ion model results showing differences in capacity for intermittent loads with periods of 10s and 1000s. Loss of capacity is a bigger effect than recovery.

on-time, which allows concentration gradients to become more pronounced, resulting in larger voltage drops due to concentration overpotential. (See Appendix C for an explanation of concentration overpotential.) The larger voltage drops in turn cause the battery to reach its cutoff voltage earlier.

The characteristics displayed in Figure 3.7, Figure 3.8, and Figure 3.9 mean that minimizing energy per operation may not maximize computations per battery life. For example, suppose a mobile system has a dynamic power profile that is cyclic, having periods of activity with a high peak power followed by idle periods of low power. If one has a choice between a 20% reduction in the energy per cycle by reducing the idle power and a 20% reduction in the energy per cycle by reducing the active power, the average power is reduced by 20% in both cases. If the battery capacity were constant as is commonly assumed, one would expect that the battery life would increase by a factor of 1/(1-20%) = 1.25 for both cases. But because the capacity is determined by the peak power, the battery life will be increased more by reducing the active power than by reducing idle power. Not only will the average power be reduced but the capacity available will be increased. Hence, once all the subsystems that can be put into idle mode are put into idle mode, one should focus on reducing the power during the active time rather than focus on reducing the power during the idle time.

A second example is if one has a choice between reducing the active time and the active power by some factor. Both will result in the same decrease in the average power. But again, reducing the active power will result in a bigger increase in battery life. This means that the focus should be on reducing peak power rather than reducing duty cycle.

For a more concrete example of each method of reducing average power, consider the dynamic power profile as shown in Figure 3.10. The average power, P_{ave}, is equal to $(P_{active} \times t_{active} + P_{idle} \times t_{idle})/t_{cycle}$. To reduce the average power P_{ave}, the active power can be reduced (A), the idle power can be reduced (B), or the active duty cycle can be

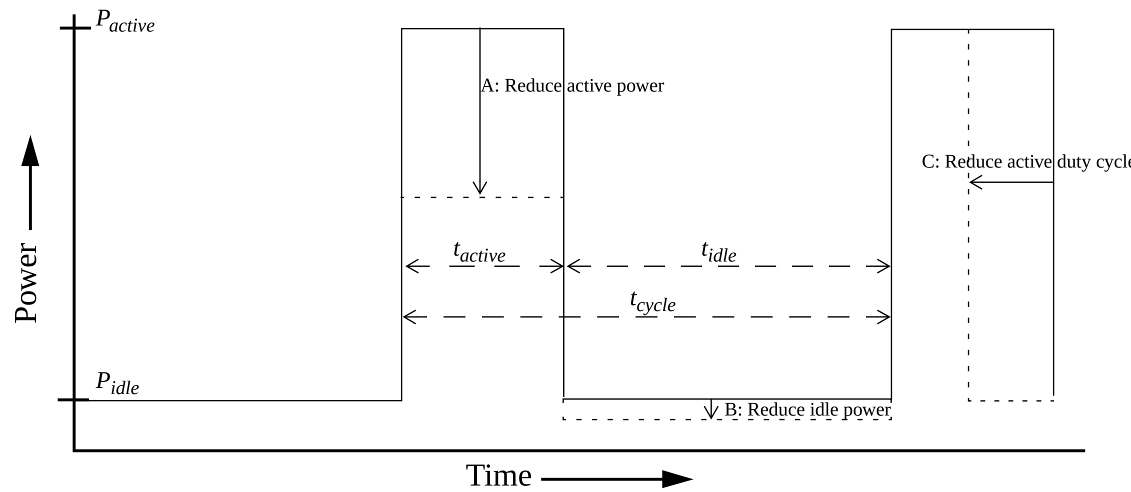


Figure 3.10 Dynamic power profile example. Modifications A, B, and C reduce the average power.

reduced (C). Table 3.2 shows the results from Doyle's model for the waveform of Figure 3.10. The waveform was simulated for three different values of initial average power, and the desired reduction in average power for each case was 20%. As expected, reducing active power (A) results in the greatest increase in battery life when the peak power is large. Reducing idle power (B) always results in the least increase in battery life. Reducing the duty cycle (C) always does better than reducing the idle power and does as well as reducing peak power only for the lowest value of peak power. This can be explained for the similar reasons as the loads with 10 second periods in Figure 3.9 having a larger capacity than the loads with 1000 second periods: Reducing the duty cycle means that there is less time for the concentration gradients to become pronounced, and consequently the concentration overpotential. But when the peak power is larger, reducing the duty cycle does not increase the battery life by as much as reducing the active power.

The column labeled "% difference from expected" refers to difference between the simulated battery life of the modification and what would be expected given the initial battery life and the factor by which the power was reduced. For example, the initial battery life of

Table 3.2. Doyle's model results for waveform of Figure 3.10.

Waveform modification	Duty Cycle, t_{active}/t_{cycle}, %	Peak power, W/kg	Idle power, W/kg	Average power, W/kg	Battery life, minutes	% difference from expected
none	20	300	75	120	51	--
A	20	180	75	96	83	+30
B	20	300	45	96	67	+5
C	9.3	300	75	96	68	+7
none	20	200	50	80	87	--
A	20	120	50	64	132	+21
B	20	200	30	64	117	+8
C	9.3	200	50	64	118	+9
none	20	100	25	40	202	--
A	20	60	25	32	268	+6
B	20	100	15	32	253	0
C	9.3	100	25	32	268	+6

the waveform with the 300 W/kg peak power is 51 minutes. Because the average power for each of the modifications is 80% of the initial waveform, one would expect the battery life for them to be 51/0.8 = 64 minutes. But this ignores the non-ideal capacity effects.

In practice, one usually does not have an initial battery life as a starting point. Typically the battery life is estimated using the rated capacity of the cell, which is measured at a very low rate discharge of 10-20 hours. For loads with large peak values, the estimate obtained using the rated capacity can be too large. Estimating the battery life using the capacity at the peak power provides a lower bound. Furthermore, it is often a closer estimate than one obtained using the rated capacity. Table 3.3 shows estimates of the battery life for the example of Figure 3.10 using the rated capacity and capacity at the peak power of the load.

Table 3.3: Estimates of battery life using rated capacity and capacity at peak power.

Waveform modification	Peak power, W/kg	Average power, W/kg	Battery life from simulation, minutes	Estimated battery life using rated capacity of 151Wh/kg, minutes	Difference from simulated, %	Estimated battery life using capacity at peak power, minutes	Difference from simulated, %
none	300	120	51	76	+48	45	-12
A	180	96	83	94	+14	74	-11
B	300	96	67	94	+41	56	-16
C	300	96	68	94	+39	56	-17
none	200	80	87	113	+30	85	-2
A	120	64	132	142	+7	120	-9
B	200	64	117	142	+21	106	-9
C	200	64	118	142	+20	106	-10
none	100	40	202	227	+12	200	-1
A	60	32	268	283	+6	261	-3
B	100	32	253	283	+12	250	-1
C	100	32	268	283	+6	250	-7

The rated capacity consistently overestimates the battery life, by as much as 50%. Using the capacity at the peak power, on the other hand, consistently underestimates the battery life and the magnitude of the error is much less than that of using the rated capacity.

These results may explain why the advertised battery life of the typical notebook computer is greater than what users realize in practice: Suppose the notebook manufacturer is advertising an estimated battery life rather than a measured one. If the manufacturer estimates the battery life by using the battery's rated capacity and the average power of the system, then the estimate will be too large because of the loss of capacity of the battery at higher rates. While the notebook computer manufacturers reap an advantage by advertising a longer battery life than is achievable in practice, obviously a motive to be considered, they may simply be using the rated battery capacity rather than the capacity available at the notebook's peak power.

These results motivate the following observations for mobile computers that are operated in the non-ideal region of their batteries:

- Total system power must be considered. Power-performance trade-offs made by examining a subsystem in isolation may not lead to an increase in the computations per battery life because total peak power is ignored.
- Peak power should be reduced wherever possible, which means background operations should be performed serially rather than concurrently. Serial operation is better than concurrent operation when each consumes roughly the same energy. (Note that serial operation will also tend to have lower context switching costs.)
- Reducing active energy is more important than reducing idle energy.
- Continuous behavior can be used to estimate intermittent behavior.

3.6 Summary

This chapter has introduced two non-ideal battery properties, loss of capacity and recovery. Simulation using a first-principles Li-ion model shows that recovery is not a problem for the typical loads and cells encountered in mobile computing, but loss of capacity can be. Simulation also shows that reducing the energy while the system is active can lead to bigger increases in battery life than reducing the energy while the system is idle, even if the amount of reduction in both cases is the same.

Consequently, power-performance trade-offs are an important method of extending battery life, as they offer a way to reduce peak power. Because of the non-ideal battery properties, an analysis of a power-performance trade-off must look not only at the total energy consumed but at the peak power of the system. Simulation of the whole system including the battery is an option, but for quickly comparing trade-offs and building intuition about them, an analytical approach is preferable.

Given that recovery is not a large effect in comparison to loss of capacity, a model for constant loads will be a good approximation for both variable or constant loads if the power used to calculate the capacity is the peak power and if the energy used while the system is idle is accounted for. Frequency and duty cycle are unimportant in terms of recovered capacity, although the duty cycle is needed to calculate the percentage of energy per cycle consumed by idle time. With this knowledge, Peukert's equation can be used to include battery behavior when formulating analytical solutions to the power-performance trade-offs. The next chapter will use Peukert's equation to examine CPU speed-setting.

Chapter 4

A system approach to CPU speed-setting

Slow and steady wins the race.
--from "The Tortoise and the Hare"

As the previous chapter showed, peak power is a better indicator of battery capacity than average power is. Consequently, analyzing the case where a mobile system is continuously active offers useful insights into the problem of CPU speed-setting. Furthermore, since recovery is not a factor, the continuous behavior can be used to study systems which would typically be used intermittently.

While this does not reflect the behavior of mobile systems in actual use, it allows us to estimate the computations per discharge in actual use because of the peak-power limited behavior of the batteries as described in the previous section. Since there is a region of operation where peak power is a stronger indicator of the battery's capacity than average power, one can obtain an estimate of the number of computations that can be completed in a discharge by looking at continuous discharges. The number of computations completed when the system is used intermittently is the continuous number times the ratio of active time energy to the total energy in an active-idle cycle. This estimate is better than the estimate obtained by dividing the ideal capacity by the average power of the cycle. Furthermore, studying continuous operation allows us to focus on the three factors discussed in the Chapter 1--battery capacity as a function of system power, system power as a function of CPU frequency, and application performance as a function of CPU frequency--without having to consider the variables of idle/active duty cycle and idle power.

This chapter develops an analytical expression for the normalized computations completed per discharge, called the "*work ratio*," using Peukert's formula for the battery capacity and assuming an ideal performance speed-up. The chapter then shows the measured results of discharges using a variety of systems, verifying the predictions of the work ratio. After that, the chapter relaxes the assumption about ideal performance speed-up and delves into complications due to the memory hierarchy. The results show that batteries caused a difference of 10%-40% from the behavior expected using ideal assumptions, and the memory hierarchy up to an additional 40% difference. The chapter concludes with an estimate of the system power if the CPU employed a variable-voltage supply.

4.1 Work ratio

This section develops an analytical expression for the normalized computations per discharge for a simple case. This expression will show that the assumptions used in developing the CPU speed-setting policies described in Chapter 2 are faulty. Only one of the assumptions will be changed, the assumption of constant battery capacity.

As described previously, Peukert's formula for the capacity of a battery is $Q = k/I^{\alpha}$, where Q is the capacity, k is a constant, I is the load current, and α has a value of between 0.2 and 0.7. For modern batteries, α is typically around 0.2. The system power P will be assumed to be a linear function of the CPU frequency f, or $P = S + CV^2f$, where S is the portion of power independent of the CPU frequency and CV^2f is the dependent portion. The final assumption in deriving the analytical expression is that the performance is proportional to the CPU frequency.

Using these assumptions, the variable ρ is defined to be $S/(S+CV^2F)$, where F is the slowest frequency at which the system will be operated, and the normalized speed, n, is defined as f/F. ρ is a measure of how much of the system power is consumed by subsystems other than the CPU. If ρ is small, then the power of the system is nearly proportional to the CPU frequency f. But if ρ is near unity, then the power of the system is almost independent of the CPU frequency.

Using these definitions of ρ and n, the number of computations completed per discharge, normalized to the number completed at the slowest frequency F, can be shown to be

$$W_n / W = n\left(\frac{1}{\rho + n(1-\rho)}\right)^{1+\alpha}$$

(See Appendix A for the full derivation.) This expression for the normalized computations per discharge is called the "*work ratio*."

To understand what the work ratio tells us about the system, consider the case where ρ is high, i.e. where the power of the system is almost independent of the CPU frequency. Figure 4.1 shows the work ratio's prediction for a system with ρ = 0.8. The difference between the work ratio for an ideal battery (α = 0.0) and a non-ideal battery (α = 0.2) is about 10% at n = 5 and about 20% at n = 10. In both cases, however, the value of the work ratio increases as the frequency increases, at least for this range of normalized frequency. The system completes more work as the CPU speed is increased. Hence, for systems with a high value of ρ, it is best to set the CPU speed as high as possible, whether the batteries are ideal or not.

On the other hand, consider the case where ρ is small, where the power of the system is nearly proportional to CPU frequency. Figure 4.2 shows the work ratio predictions for ρ = 0.3. When the battery is ideal, the system should be run as fast as possible, although the increase in the work ratio as the frequency is increased is less than for high ρ systems.

When the battery is non-ideal, however, increasing the CPU speed beyond a certain point causes the work ratio to decrease. The difference between the ideal and non-ideal predictions for $\rho = 0.3$ is about 25% for $n = 5$ and 35% for $n = 10$. The important point here, though, is that running as fast as possible, which is what should be done if the batteries are ideal, will cause the computations completed in a discharge to decrease when the batteries are non-ideal, even though the energy per operation decreases. For the non-ideal battery, low-ρ case, there is a speed that maximizes the amount of work completed in a discharge. Operating at higher speeds is detrimental.

To consider the general case for all values of n and ρ requires a 3-D plot. Figure 4.3 shows the contours of the 3-D curve for the work ratio for $\alpha = 0.2$, $0 < n < 10$, and $0 < \rho < 1$. The x-axis is the speed-up n, and the y-axis is ρ. The contours are the value of the work ratio.

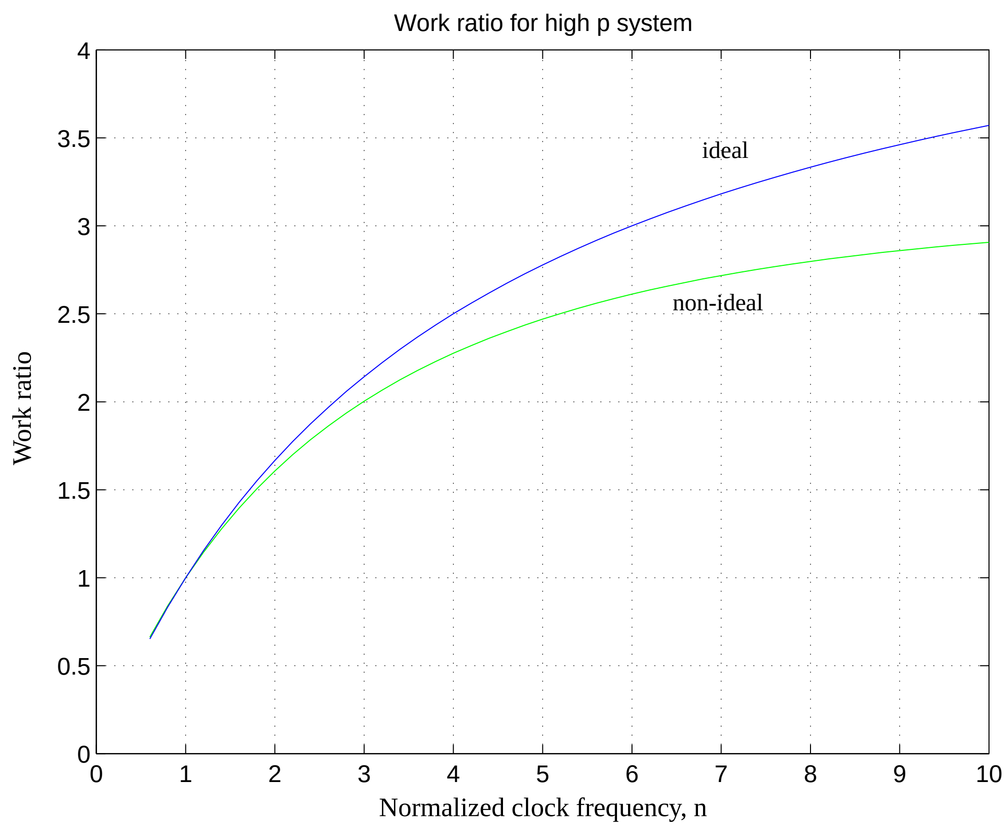


Figure 4.1 Work ratio predictions for ideal ($\alpha = 0.0$) and non-ideal ($\alpha = 0.2$) batteries for $\rho = 0.8$

The dashed line shows the maximum value of the work ratio for a given ρ. There are two regions in the plot, one where increasing the clock frequency from $n = 1$ is initially beneficial and one where decreasing it from $n = 1$ is initially beneficial. The two regions are separated by the dashed line. The region where it is initially beneficial to increase the clock frequency is above the line; the region where it is initially beneficial to decrease the clock frequency is below the line. Only in the two ideal cases, ρ = 0 or 1, is it always better to decrease (ρ = 0) or increase (ρ = 1) the clock frequency. For all other values of ρ, a maximum occurs at $n = \rho/\alpha(1-\rho)$ (see Appendix B).

For a given system, changing the clock frequency corresponds to a horizontal movement from (1, ρ) to (n, ρ). ρ remains constant because it was defined using only the initial CPU

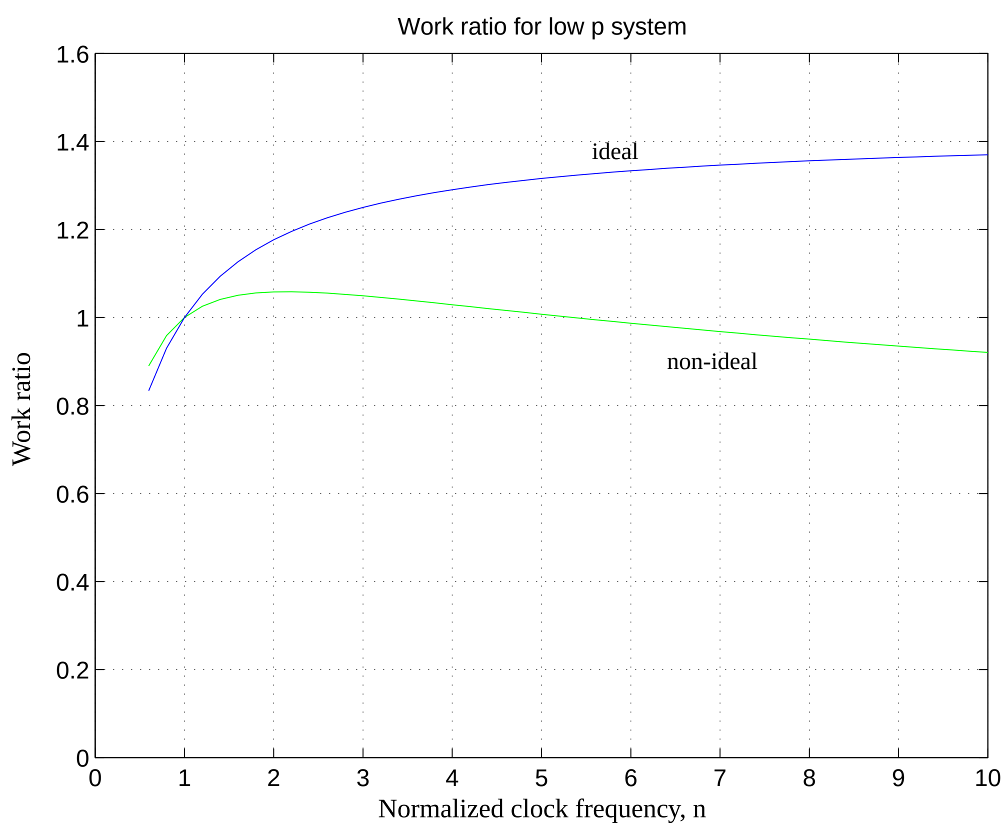


Figure 4.2 Work ratio predictions for ideal (α = 0.0) and non-ideal (α = 0.2) batteries for ρ = 0.3

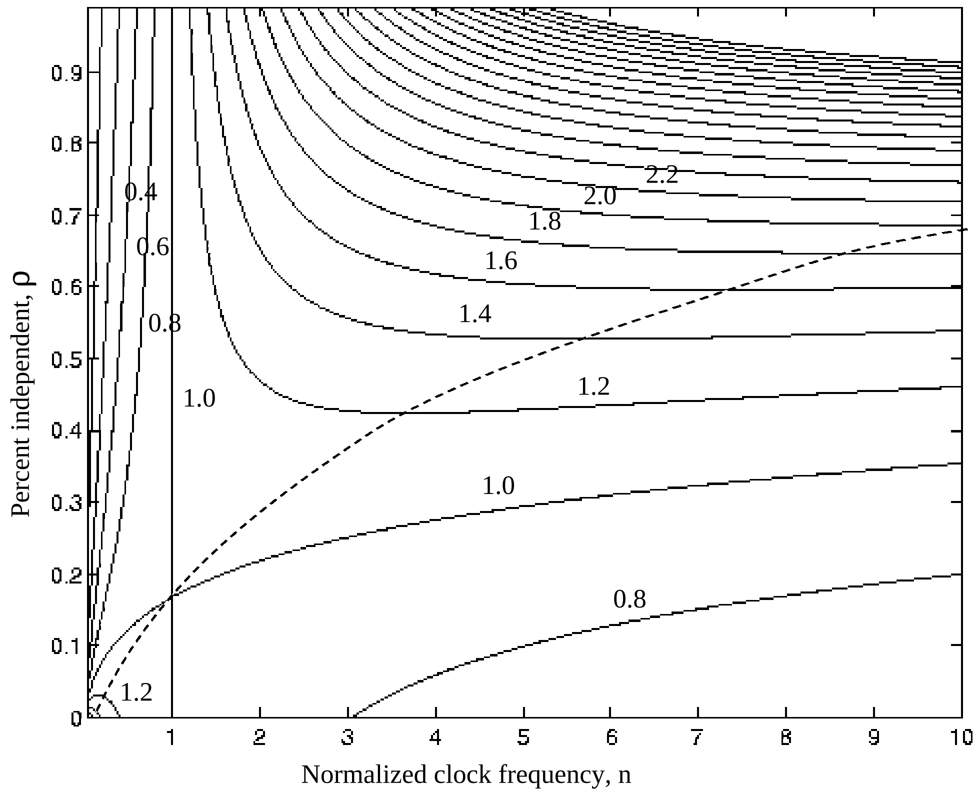


Figure 4.3 Work ratio contours, α = 0.2

speed *F.* In order to complete the most computations per battery life, *n* should be set to the value which maximizes the work ratio. For $n > 1$, if this new, faster system is used to define a ρ', then ρ' < ρ. If $n = \rho/[\alpha(1-\rho)]$, then (1, ρ') is the point where the 1.0 contours cross. (See Appendix A for a full derivation.) At this value of ρ, the CPU frequency is optimal, because any change in *n* will result in a decrease in the work ratio.

The VuMan 2 wearable computer will serve as an example of how the contour plot can be used. The power of the VuMan 2 was measured to be (0.84 + 0.015 x CPU frequency) Watts. The minimum operating frequency of the design is 12 MHz and the maximum is 24 MHz. Suppose the current design operates at 12 MHz and a designer would like to know if would be better to operate at some other frequency. At 12 MHz, then ρ = 0.84/(0.84 + 1.05 x 12) = 0.82. At ρ = 0.82 and n = 1 in Figure 4.3, decreasing the frequency decreases the work ratio, and increasing the frequency increases the work ratio. The designer would then decide to increase the clock frequency. In this case, the designer would be limited by the maximum frequency of 24 MHz. 24 MHz corresponds to $n = 2$, with the work ratio being approximately 1.6, or a 60% increase in the number of iterations executed during a battery life.

Suppose the designer, in order to take advantage of the work ratio increases beyond 24 MHz, wanted to increase the maximum frequency. The peak work ratio occurs at $n = \rho/[(1-\rho)\alpha] = 0.82/(0.18 \times 0.2) = 22.8$, or at 274 MHz. The system should not be operated beyond this point.

Suppose, however, that the design were modified to lower the static portion of the power from 0.84 W to 0.10 W. The design still operates at 12 MHz, so $\rho = 0.10/(0.10 + 0.015 \times 12) = 0.35$. Figure 4.4 shows a close-up of the contour plot around $\rho = 0.35$, with an ‘x’ marking the current design and a line showing the possible increase in the work ratio by increasing the clock speed. The maximum occurs near $n = 2.7$, with the work ratio approximately 1.10. The close-up shows that the work ratio is very flat from $n = 2$ to $n = 4$ for $\rho = 0.35$. While the maximum occurs at $n = 2.7$, little could be gained by increasing the maximum frequency of the design beyond 24 MHz.

4.2 Work ratio results

To verify the work ratio for large values of ρ, a set of experiments was performed with several mobile systems. Each system ran an application in a continuous loop, and the number of iterations of that loop completed in a discharge cycle were counted over a range of CPU frequencies. The systems used were the Navigator 1, a wearable computer developed at CMU [65], and the Itsy, a hand-held system from Compaq’s Western Research Lab [71]. To verify the work ratio for small values of ρ, a clock circuit was constructed with a large capacitive load. By adding resistors, the value of ρ could be varied over a broad range.

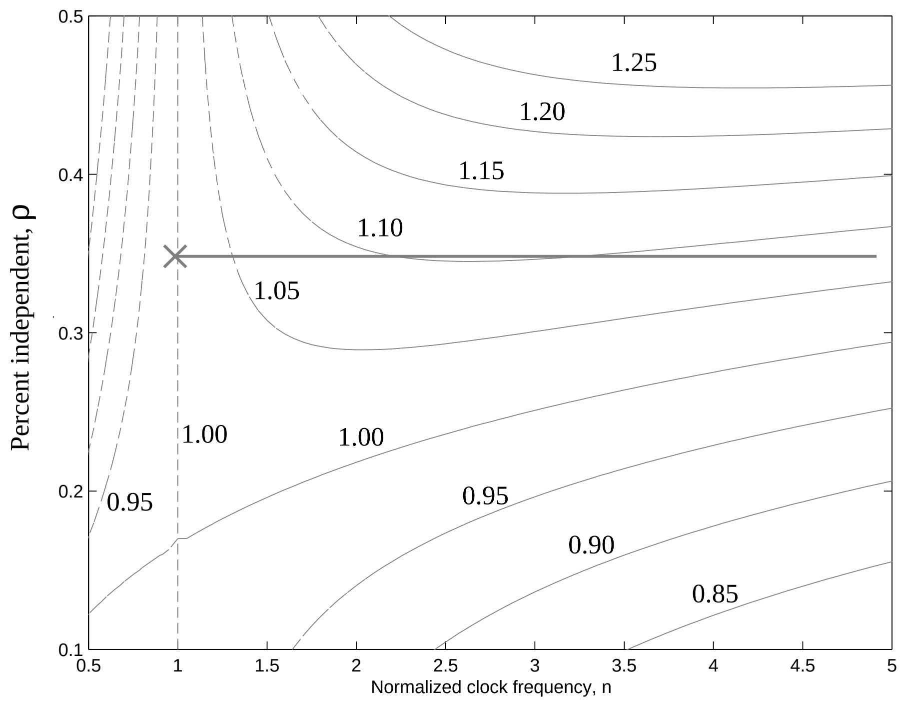


Figure 4.4 Close-up of work ratios around ρ = 0.35, showing the maximum possible increase for that ρ

The Navigator 1 is based on the Intel 80386 processor, runs the Mach operating system, and was designed to be a campus tour guide. For the experiments with the Navigator 1, the spinning hard drive was replaced with bootable EEPROM, the operating system was changed to DOS to fit into the EEPROM, and the application was changed to be a simulation of the VuMan 2 wearable computer. The system was designed to run at 25 MHz, but by changing the CPU's clock crystal, the system could run from 7.2 MHz up to 32 MHz. Its ρ value was measured to be 0.81.

4.2.1 Navigator 1 results

The Navigator 1 results shown in Figure 4.5 and Figure 4.6 confirm the work ratio predictions for the high ρ case. At a normalized clock frequency of 4.5, the Navigator 1 shows about a 10% difference from ideal for both Canon and Sony Li-ion batteries, matching the predictions of the work ratio very well. That it matches for sets of batteries from two different manufacturing processes is also encouraging, as it means that the effect is not due to the properties of a particular manufacturer's product.

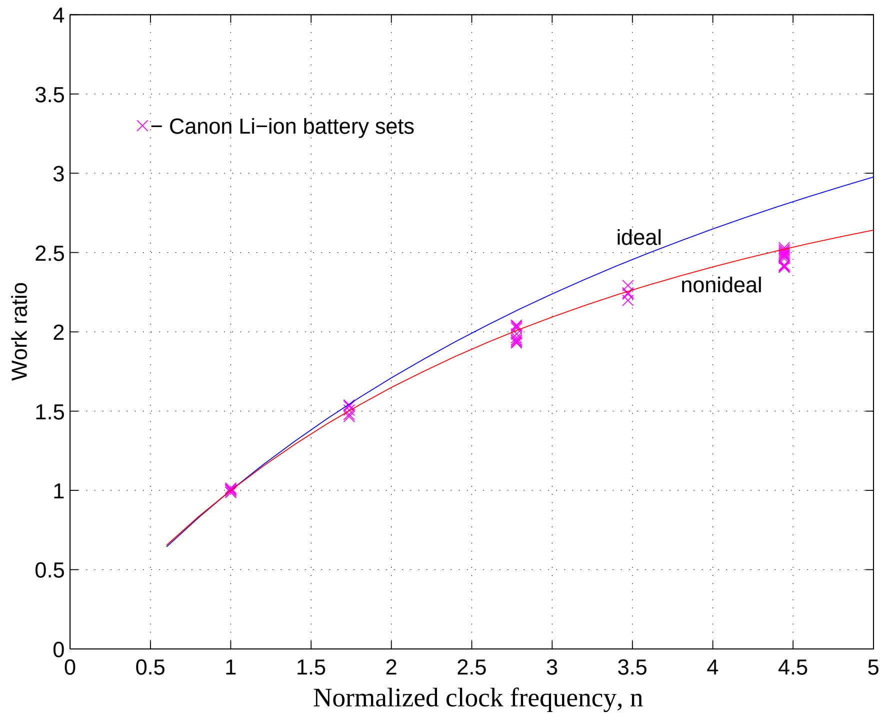


Figure 4.5 Work ratio ideal prediction, non-ideal prediction, and experimental results with Navigator 1 and Canon BP-911 Li-ion batteries.

4.2.2 Simple circuit results

None of the computer systems available had a low value of ρ. To test the work ratio predictions for the low-ρ case, a clock circuit with a large capacitive load was constructed. Because of the large capacitive load, its power consumption was proportional to fCV^2. By adding resistors in parallel the ρ of the simple circuit could be varied over most of the range from zero to unity. Figure 4.7 shows the measured results with the low ρ system ($\rho = 0.14$) and the predictions of the work ratio for both ideal and non-ideal batteries.

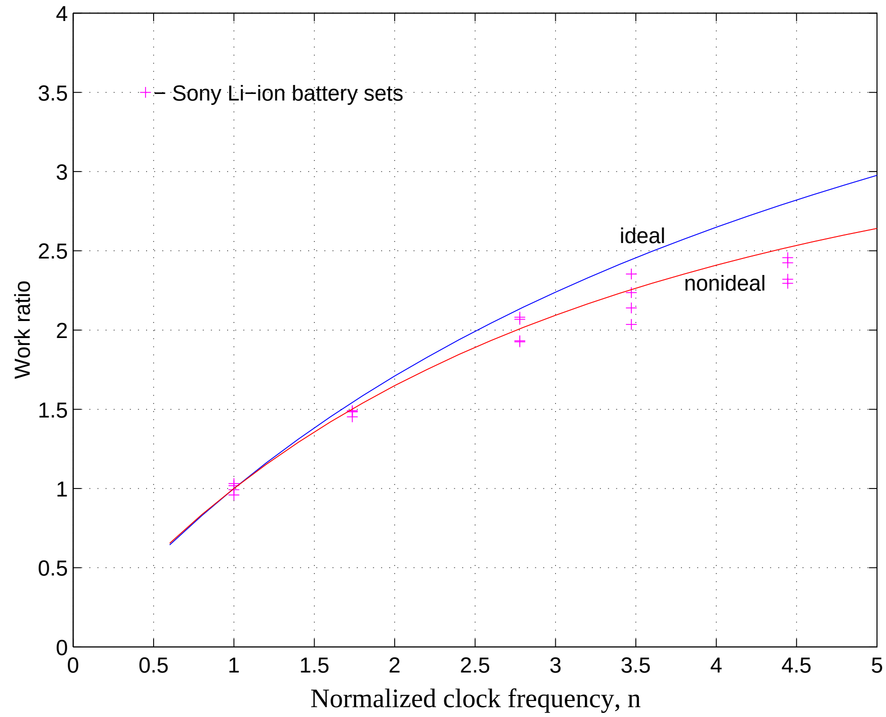


Figure 4.6 Work ratio ideal prediction, non-ideal prediction, and experimental results with Navigator 1 and Sony LIP-2000 Li-ion batteries.

Again the non-ideal battery work ratio prediction matches the measured results quite well, showing a decrease of about 30%.

For the low-ρ case, setting the CPU speed too high will decrease the number of computations that can be completed in a battery life. This runs counter to intuition and to the previous work. According to both, for a constant-voltage system, running as fast as possible should at worst perform the same work per discharge as running at slower speeds, and in most circumstances should perform more work per discharge. But because of the loss of battery capacity, running as fast as possible decreases the work per discharge in the low ρ case. Thus non-ideal battery behavior must be taken into account when setting the CPU speed, especially when most of the power of the system is consumed by the CPU.

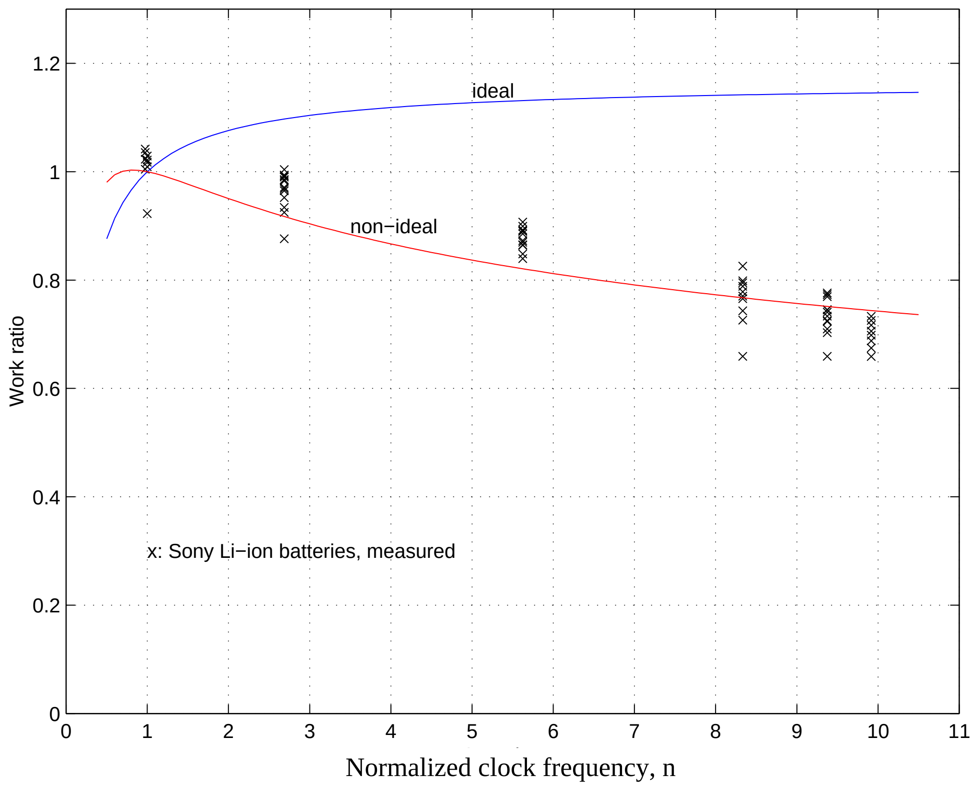


Figure 4.7 Work ratio ideal prediction, non-ideal prediction, and experimental results with low-ρ circuit and Sony LIP-2000 Li-ion batteries.

4.3 Non-ideal performance effects

The previous section showed the impact of non-ideal battery behavior. This section reexamines the assumptions about performance speed-up as the CPU clock frequency is changed. The ideal system behavior assumed by the previous work is that performance scales with CPU frequency, which ignores the memory hierarchy. It turns out that memory system behavior can also reduce the computations per battery life. The results in this section show that if the memory bandwidth does not increase at the same rate as the CPU frequency, then the performance will not scale with frequency for programs with poor cache behavior. The data in this section were generated with the Itsy. Before describing the data, it is necessary to give the reader some background about the Itsy system.

4.3.1 Details of changing the clock speed on the Itsy

The Itsy is based on the StrongARM SA-1100 [14] and runs the Linux operating system [60]. Two calls were added to the Linux kernel for changing the CPU frequency based upon the clocking scheme for the StrongARM, which is shown in Figure 4.8. During normal operation, the CPU core runs at the primary clock frequency and the memory controller runs at half the primary clock frequency. On a cache miss, the CPU core clock switches to half the primary clock frequency so that the core and the memory controller run synchronously. The first kernel call forces the CPU core clock to run at half the primary frequency at all times, referred to as running with clock switching disabled (i.e. the core clock cannot switch between the primary clock frequency and half the clock frequency). The second kernel call selects the value of the primary clock frequency, setting it to one of eleven possible values between 59 and 206 MHz. The overhead of the first call is only a few CPU cycles, while the overhead of the second call is approximately 150 μs while the on-chip phase-locked loop acquires the new frequency [14]. The combination of the two calls allows the CPU core frequency to vary from 29.5 to 206 MHz.

The kernel call for running with clock switching disabled writes to register 15 of coprocessor 15 of the SA-1100 as described in the SA-1100 Technical Reference Manual [14], requiring less than 10 instructions. The call for changing the primary clock frequency is considerably more involved. Because many of the peripheral devices experience an inter-

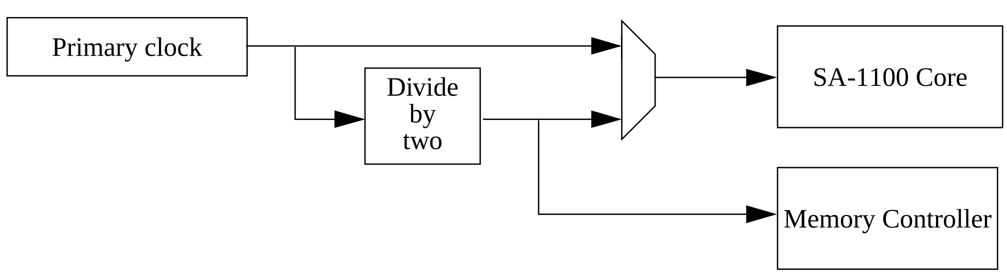


Figure 4.8 Clocking scheme for the StrongARM SA-1100

ruption of service while the phase locked loop locks on to the new frequency, the call first ensures that these devices are not in use by calling the power management suspend routine, which takes care of all the devices except for the memory controller. If the CPU clock is being switched to a higher frequency, the memory controller is updated before changing the clock frequency; if the clock is being switched to a lower frequency, then the memory controller is updated after changing the clock frequency. In both cases, the FLASH memory configuration data is changed while the CPU is executing instructions from DRAM but the DRAM memory configuration data is changed while the CPU is executing instructions from FLASH. This requires that the instructions for changing the DRAM memory configuration be compiled separately from the rest of the Linux kernel and burned into FLASH separately. It was found that the clock frequency could not be changed reliably for certain combinations of initial and final clock frequencies without executing from FLASH while changing the DRAM memory configuration. After the memory controller values have been changed, the peripheral devices are awakened by a call to the power management wake-up routine. The pertinent portions of the source code for the clock device driver (`clock.c, clock.h, clock-dram-setup.S`) are included in Appendix E, as is the source code for the instructions for changing the DRAM memory configuration (`clock-patch.S`).

4.3.2 Experimental set up

The code chosen for the experiments was the Itsy's MPEG video player. The justification for this choice is twofold. First, a video player is expected to be one of the typical applications for a mobile computer. Second, the code should adequately exercise the memory hierarchy and operating system for reasons similar to those given by Agarwal [1]. Measuring the energy of an application that fits into the cache or that uses no operating system resources would not be representative of a mobile computer capable of running a variety of applications. Uhlig et al. describe how the programs in the SPEC suite have much lower cache miss ratios than real applications, and the MPEG player is one of the applications they chose for their alternative benchmark suite with more realistic cache and system behavior [70].

In the battery life measurements, the MPEG player ran in a continuous loop. Each run of the MPEG player was followed by decompression and manipulation of a text file in order to clear the caches of the MPEG instructions and data. The MPEG player accounted for more than 90% of the run-time of the loop, however, so it was the dominant energy component. The loop ran without any idle time.

All current measurements were made using a Fluke 45 true RMS, digital multimeter. The Itsy has several 20mΩ precision resistors for measuring currents on each of its power supplies. The Fluke makes 10 readings per second, sufficient speed given that the time constants of the electrochemical reactions are such that the battery sees only the average for load changes that occur at frequencies greater than about 1 Hz [33].

4.3.3 Itsy results

This section describes the measurements of performance versus frequency, power versus frequency, and iterations per discharge of the Itsy running the MPEG player.

Figure 4.9 shows the ideal and measured computations per discharge for the Itsy powered by AAA batteries. The measured values are considerably less than ideal. One might suspect that the Itsy is a low-ρ system, but the power versus frequency measurements of Figure 4.10 show the ρ to be 0.65, definitely not a low ρ system. This value of ρ was calculated using a least squares fit of the data in Figure 4.10. For this data, $\mathit{power} = 0.0047 \times \mathit{frequency} + 0.26$. Using the definition of ρ, then $\rho = 0.26/(0.0047 \times 29.5 + 0.26) = 0.65$. For this value of ρ one would expect the measured results to differ from the ideal by about 20% at a speed-up of 7. The reason for the difference turns out to be the performance behavior of the memory hierarchy.

Upon closer examination, the assumption of ideal performance speed-up does not hold for the Itsy and the application it was running. Figure 4.11 shows the ideal and measured performance versus frequency for the MPEG loop. Both are normalized to the performance at the slowest speed. The measured performance is nearly ideal until 100 MHz, where there is a breakpoint. From 100 MHz to the maximum frequency of 206 MHz, the measured performance diverges from the ideal.

The Unix tool *gprof* [32] was used to examine the performance of the two functions that take the most execution time. One has almost ideal speed-up whereas the other is far from ideal. Figure 4.12 shows the normalized performance versus the CPU frequency for these two functions. The points show the measured performance, the solid line shows the expected performance given the change in clock speed, and the dashed line shows the nor-

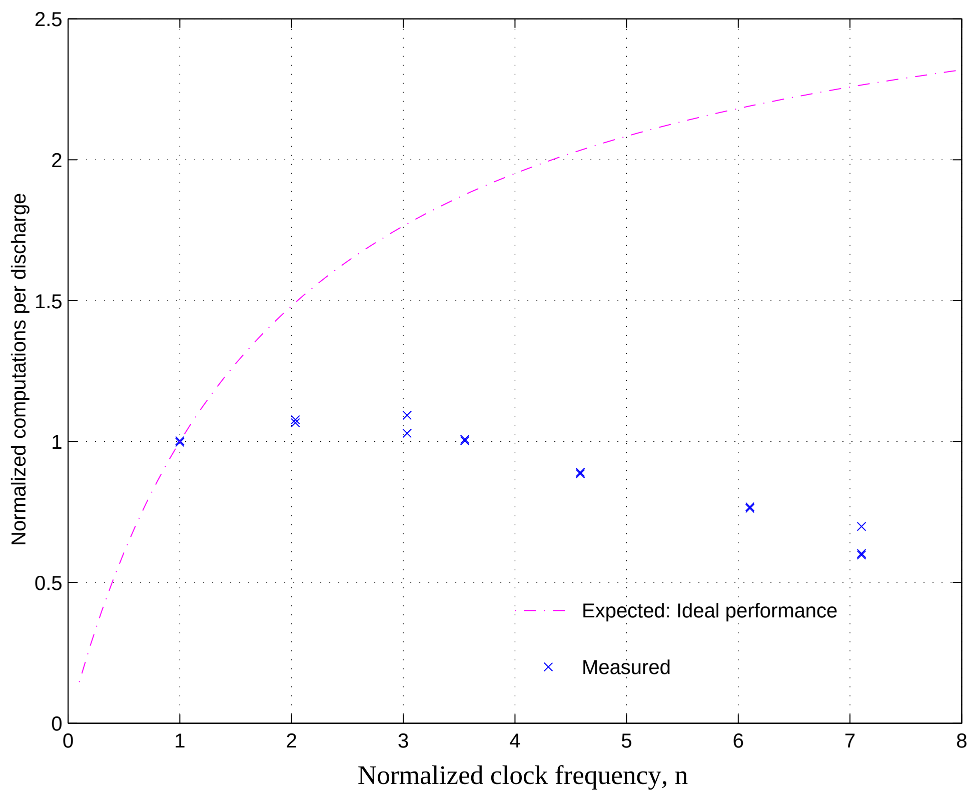


Figure 4.9 Expected and measured results with AAA batteries

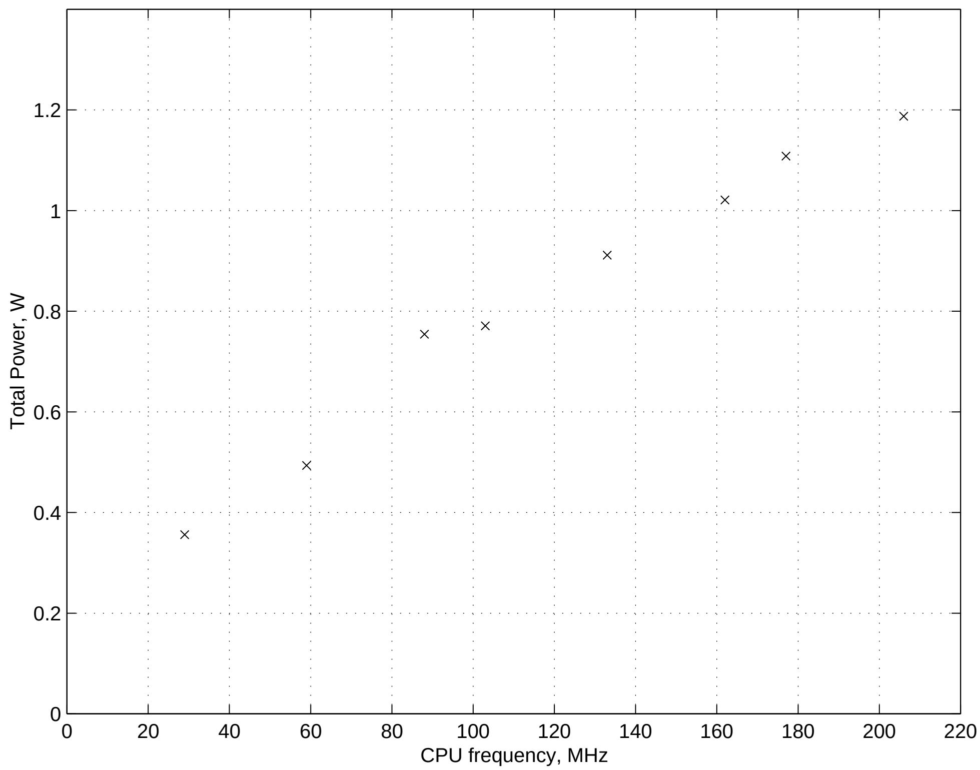


Figure 4.10 Power vs. frequency

malized, measured main memory bandwidth. The routine *mpeg_j_rev_dct* has the expected performance speed-up over nearly the entire range of CPU frequencies. But the other routine, *GrayDitherImage*, shows little increase in performance after 133 MHz, where the main memory bandwidth becomes limited by the speed of the memory chips. Below 133 MHz the memory bandwidth is determined by the minimum number of cycles per access required by the SA-1100's memory controller, while above 133 MHz the memory bandwidth is determined by the access time of the memory chips. So, as the CPU frequency increases above 133 MHz, accesses to main memory take more CPU cycles. The performance of *GrayDitherImage* tracks the memory bandwidth rather than the CPU frequency, limiting the performance speed-up of the program overall.

Figure 4.13 shows the expected iterations per discharge calculated from the power and performance measurements from Figure 4.10 and Figure 4.11, and the iterations per dis-

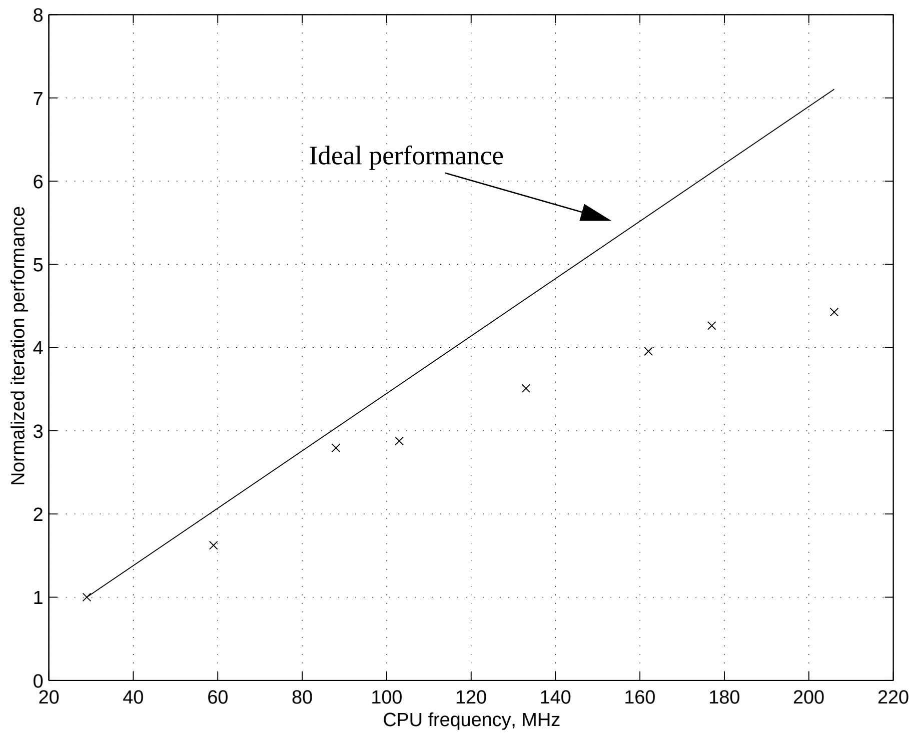


Figure 4.11 Performance vs. frequency of MPEG loop

charge if the performance speed-up were ideal. Both curves assume ideal batteries and are normalized to the slowest speed, 29.5 MHz. The iterations per discharge with a non-ideal performance speed-up differs from the ideal iterations per discharge by 40%. This prediction was confirmed with actual battery discharges while running the MPEG player using Sanyo UF-510 and UF-310 Li-ion cells [62][63], shown in Figure 4.14, along with the expected iterations per discharge curve from Figure 4.13. These cells are nearly ideal over the range of power of the Itsy. Consequently, the measured iterations per discharge agree with the expected curve quite well, although the measured points begin to fall below the expected at the two highest frequencies. But the major feature of Figure 4.14 is that ideal assumptions about performance versus frequency can be quite misleading.

But Figure 4.15 shows that ideal assumptions can be even more misleading when the battery is non-ideal over the range of the power of the system. These results were collected

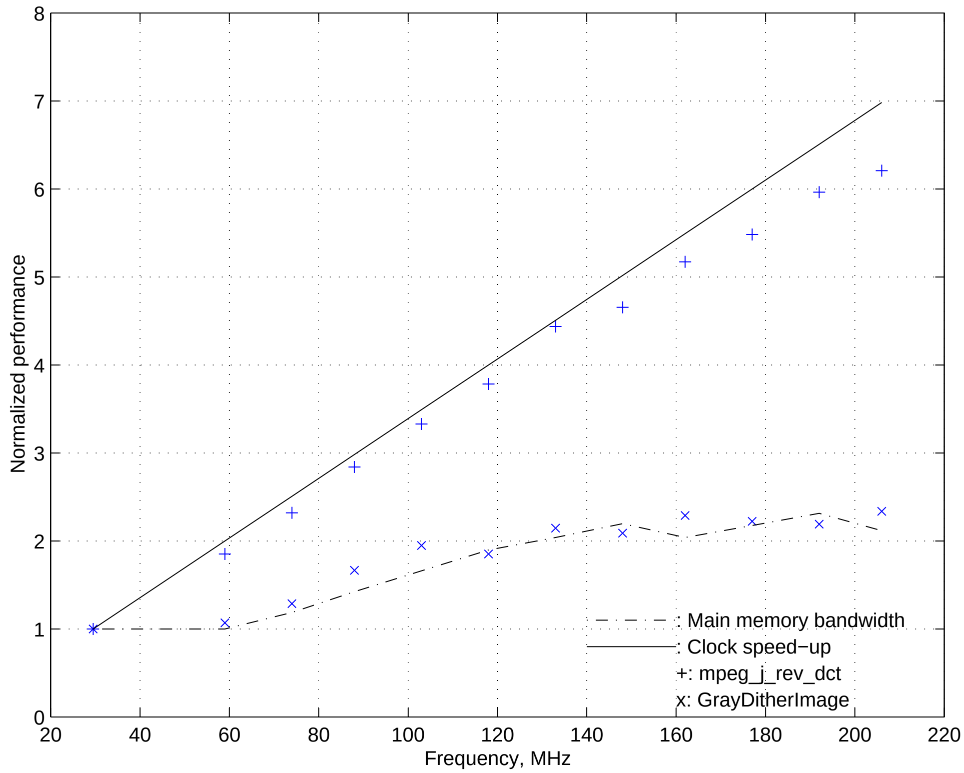


Figure 4.12 Performance versus frequency for two of the MPEG player functions, with expected speed-up and measured main memory bandwidth

using the AAA alkaline cells which the Itsy was designed to use. Because of the loss of battery capacity, the iterations per discharge decreases at higher CPU frequencies even though the energy used per iteration remains nearly constant. One might suspect that the results in this figure are due to the batteries being loaded at an unrealistic value. The "on time" of the highest frequency case is approximately 40 minutes, certainly a realistic load since the system is continuously active. For comparison, notebook computers often have battery lives of 1.5-3 hours on the ZDigit BatteryMark test, which is active less than 20% of the time, giving them a continuous "on time" of less than 40 minutes [56]. Thus one would expect to see similar results if the experiments were conducted using the notebook computers.

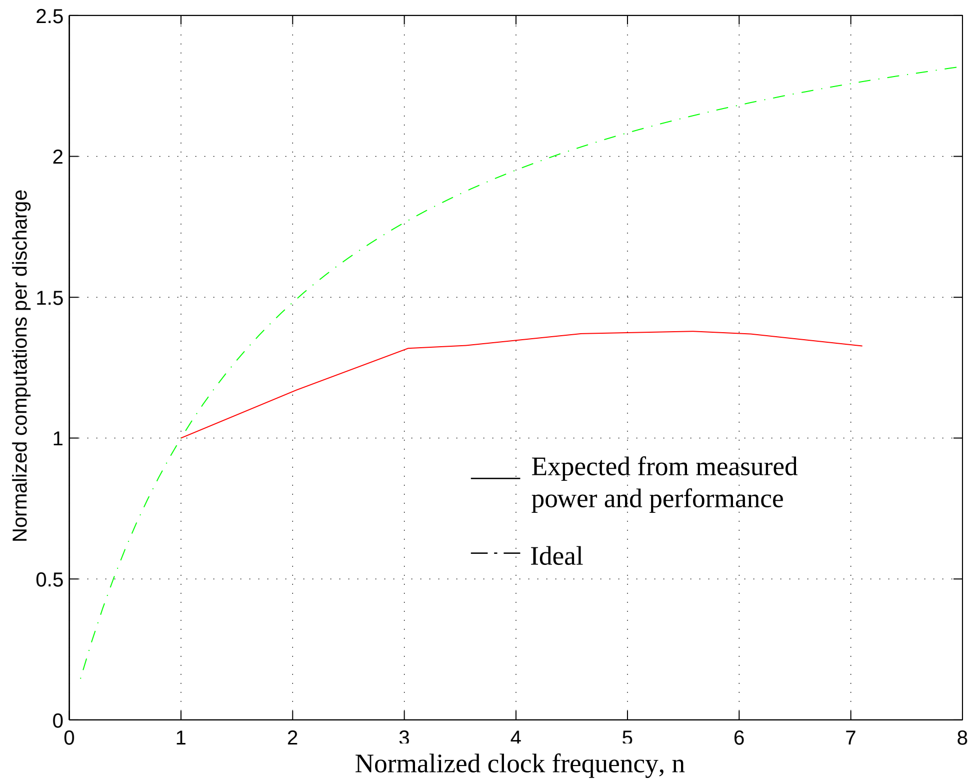


Figure 4.13 Ideal and expected computations per battery life

Figure 4.15 shows that assuming ideal behavior for performance versus frequency and battery capacity versus power can be mistaken. The measured iterations per discharge were more than 70% less than the iterations per discharge predicted by ideal behavior for both factors. Even using measured performance versus frequency, the iterations per discharged decreased as the clock frequency was increased, rather than increasing as predicted by assuming an ideal battery capacity.

4.4 Total system power with variable-voltage CPU

The Itsy results shed light on the idea of using variable voltage CPU’s for speed-setting. Obviously, the previous work on policies for variable-voltage CPU’s should be augmented to include the non-ideal battery and performance properties. But it also should include information about the total system power.

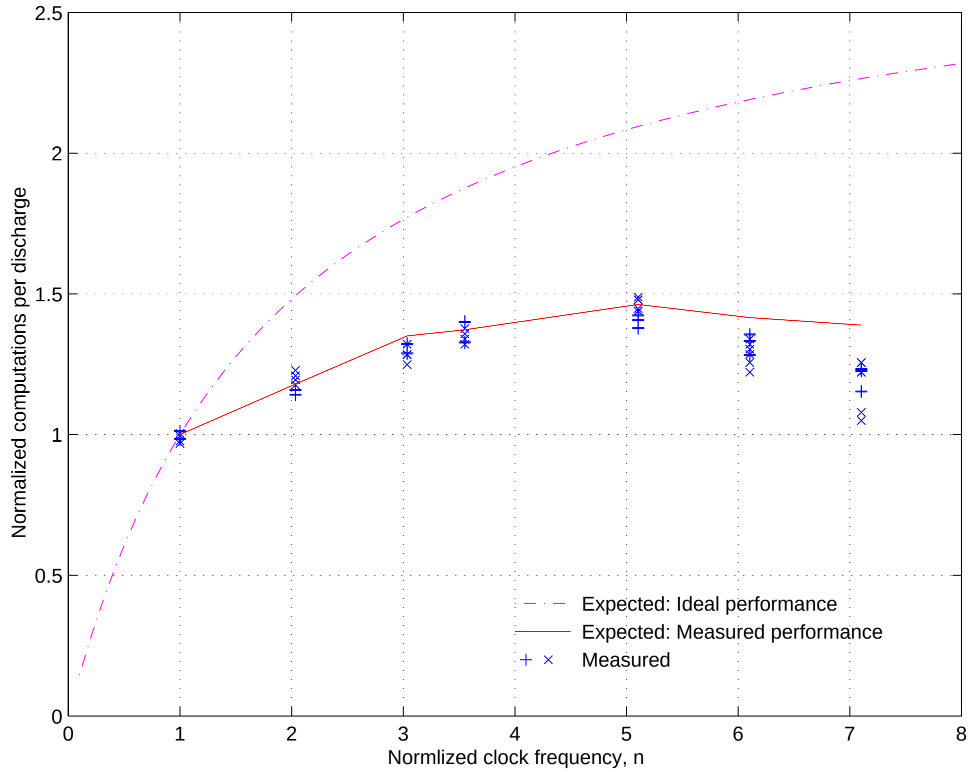


Figure 4.14 Expected and measured results with Li-ion batteries, Sanyo UF-510 and UF-310

Suppose the Itsy's SA-1100 were a variable-voltage CPU. The SA-1100 has two power planes: a 1.5V core voltage and a 3.3V voltage for the peripheral pins. Of these two, only the 1.5V plane could be varied with the CPU frequency. The 3.3V plane would have to remain fixed to be connected with the other subsystems. The points on Figure 4.16 show the total measured power of the Itsy and the power of the 1.5V and 3.3V planes versus the CPU frequency. (The measured total power is greater than the sum of the power on the two planes because of losses in the power supplies.) The lower dashed line is an estimate of the power of the 1.5V supply if it were scaled with the CPU frequency. The upper dashed line is this estimate added to the power of the 3.3V supply, which, as was stated above, must remain fixed. The upper dashed line is thus an estimate of the total power of the Itsy if the SA-1100 were a variable voltage CPU.

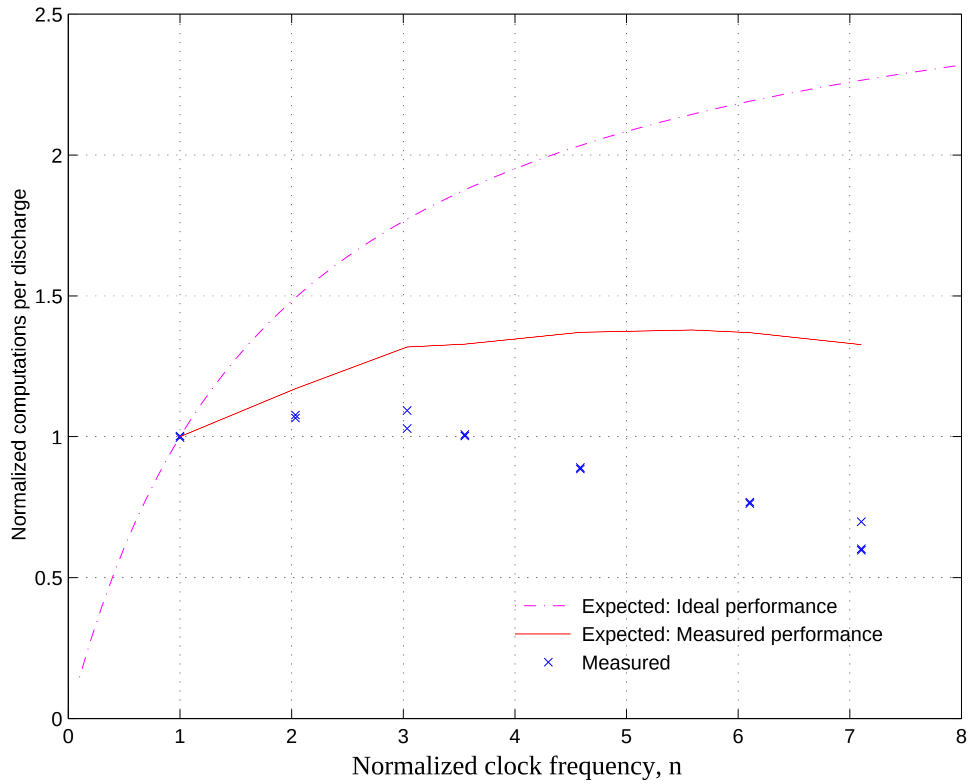


Figure 4.15 Expected and measured results with AAA batteries

The Itsy's total power in this case would be nearly linear, with a large y-intercept. It would hardly vary as s^3, where s is the normalized clock frequency, the fundamental assumption used in formulating the speed-setting policies in the previous work. Because it is nearly linear with a large y-intercept, there will be a frequency greater than zero that minimizes the energy per operation. The policies from the previous work assume that running as slowly as possible minimizes the energy per operation. But this stems from looking at only the power of the CPU. This example shows that it is necessary to look at the total system power to set the CPU speed correctly. Even with a variable-voltage CPU, the system power as a function of the normalized frequency s becomes $a_3s^3 + a_2s^2 + a_1s + a_0$ rather than simply s^3. Because of the lower order terms, the energy per operation is minimized at a frequency which is greater than the minimum operating frequency of the CPU. Conse-

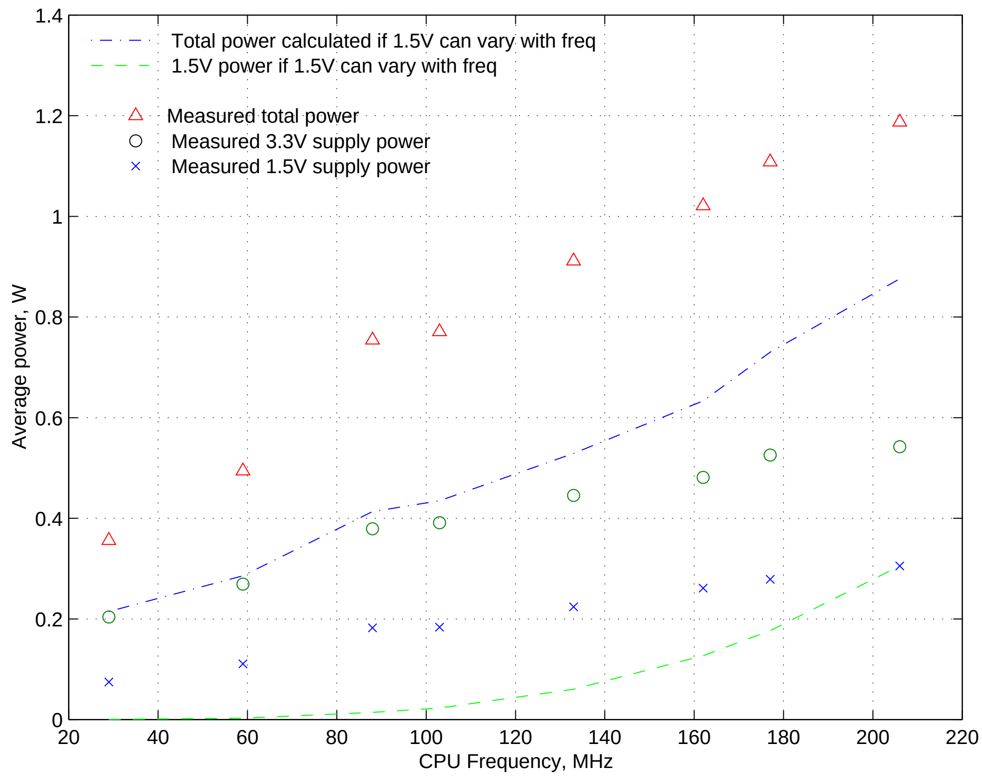


Figure 4.16 Measured total power, 3.3V and 1.5V supply power, and estimates of 1.5V supply and total power of Itsy if SA-1100 were a variable voltage CPU.

quently, a CPU speed-setting policy must take the lower order terms into account, as the next chapter will show.

4.5 Summary

This chapter has developed an analytical expression for the amount of computations completed in a discharge, called the "*work ratio*". The work ratio predicts that systems with high values of ρ should be run with the CPU speed set at maximum speed, and that systems with low values of ρ should be run at less than maximum speed. The predictions of the work ratio were verified for both high- and low-ρ systems.

The wearable computer systems developed at CMU all have ρ values ranging from 0.8-0.9. The question arises as to whether or not all mobile systems will have values of ρ in this range, and if so, will they continue to have values of ρ in this range. In other words, do all mobile systems have power composed mainly of non-CPU power, or will CPU's begin to dominate the power in the future? The likely answer is that there will be classes of mobile systems that are consistently either high ρ or low ρ due to the hardware requirements of their intended applications, and another class of systems that are at times high ρ systems and at other times low ρ systems, based upon the power management state of their subsystems. For the consistently high or low ρ systems, the CPU speed can be fixed accordingly. For the systems with mixed ρ behavior, the current power management state of each device must be taken into account when setting the CPU speed.

As developed in this chapter, the work ratio does not consider non-ideal performance behavior, specifically performance limited by bandwidth of the main memory bus. The results of discharge experiments with the Itsy show that the memory bandwidth will also affect the number of computations completed in a discharge. For the Itsy, non-ideal battery behavior caused a 10%-40% difference from ideal behavior and the non-ideal performance behavior up to an additional 40%. Together the non-ideal battery and performance behavior result in the measured values being up to a factor of four less than would be expected using ideal assumptions.

The Itsy also allows one to estimate the total system power if it had been built using a variable voltage CPU. Contrary to assumption of the previous work in policies for setting the speed of variable voltage CPU's, the system power as a function of frequency is likely to be composed cubic and lower order terms. Thus the goal of running as slowly as possible when there are no performance constraints is incorrect.

The next chapter shows how each of the non-ideal factors covered in this chapter can be used to develop a more realistic operating system policy for setting the CPU speed dynamically.

Chapter 5

Towards A General Purpose CPU Speed-setting Policy

Knowledge must become capability.
Clausewitz

The CPU speed-setting policies presented in the previous work [31][72] ignored the non-ideal behavior described in Chapter 3 and Chapter 4. This chapter describes a method to augment those policies so that they account for non-ideal battery properties, non-ideal performance speed-up, and lower order terms in the function of system power as a function of frequency. The chapter begins by outlining the goals of a CPU speed-setting policy. It then presents the policy *Past* [72] as an example of how the results from the previous chapter can be used, by changing the policy's lower bound on speed. The effect of each of the three factors on the lower bound is then covered, using simple cases to build intuition about their interactions and introducing a notation for describing a system's battery capacity, power, and performance. Since finding the lower bound on the speed for the most general case (non-ideal battery, general cubic function for system power, non-ideal performance) involves solving a fourth order equation, a more practical approach for finding the lower bound in general is given, and alternatives to it are discussed. The approach handles both the general case and the special cases discussed in the previous work as well. This chapter is more speculative than the previous chapters, as the hardware necessary to test the ideas presented is not currently available. Initial validation of the ideas, however, is provided by re-visiting the simulations of Weiser et al. [72]. The chapter concludes with a list of desirable system features for supporting a realistic CPU speed-setting policy.

It should be noted that it is assumed throughout the chapter that the CPU speed will be changed dynamically, at least several times a second. The policies and methods described are meant to be implemented within the operating system as part of its scheduling or power management operations.

5.1 Goals of a CPU speed-setting policy

As described by Weiser et al., two options for implementing a policy are to modify applications such that they provide hints to the operating system, which then sets the speed, or to have the operating system set the CPU speed based solely upon parameters of the system [72]. The drawback of the former method is that the applications must be modified, and so Weiser et al. opted for a policy based solely upon parameters of the system, as does this dissertation. However, the lower bounds on speed and desirable features for a speed-setting system that are discussed below are the same for both types of implementation.

A CPU speed-setting policy should have two goals. The first is to deliver performance which is not noticeably different from the performance when the CPU is always running at maximum speed [72]. The second is to allow the user to complete as much work per discharge as possible. These two goals determine the upper and lower bounds on the range of speeds that a speed-setting policy should employ. The upper bound is the maximum CPU operating frequency. The lower bound is the speed that maximizes the work per discharge, the *optimal frequency.* Running more slowly than the optimal frequency violates the goal of completing the most work per discharge possible. Running more quickly than the optimal frequency should only occur when required by performance expectations.

The policies described in the previous work used the CPU's minimum operating frequency as the lower bound [31][72]. One of the policies, *Past*, is shown in Figure 5.1 [72]. The

```
Past()

new_cpu_speed = old_cpu_speed
if excess_cycles > idle_cycles {
    new_cpu_speed = 1.0
    return
}
run_percent =
    run_cycles/(run_cycles+idle_cycles)
if run_percent > RUN_HIGH
    new_cpu_speed =
        new_cpu_speed + Delta
if run_percent < RUN_LOW
    new_cpu_speed =
        new_cpu_speed - Delta
if new_cpu_speed > 1.0
    new_cpu_speed = 1.0
if new_cpu_speed < min_cpu_speed
    new_cpu_speed = min_cpu_speed
return
```

Figure 5.1 Pseudo-code for speed-setting policy *Past* [72]

Past policy calculates the speed for the next time-interval based upon the run- and idle-percentage of the previous time interval, as well as work left over from the previous interval if the speed during that interval was too slow to handle all the load. (The left-over work is represented by the variable *excess_cycles*.) The size of the time interval was from 10 ms to 100ms, and the best results were obtained with the interval set to 20 ms. *Past* looked at the percent idle time of the last interval and assumed the next interval would have the same percent. Excess cycles from the previous interval were completed in the current interval, if possible. The frequency changes were smoothed by limiting how much the frequency could change from its previous value, using the empirically-determined constants RUN_HIGH, RUN_LOW, and Delta.

The important point to notice is that the lower bound on the speeds used by the policy is set solely by the minimum operating frequency of the CPU. The other policies in the previous work differed from *Past* in how they calculated the work to be completed in the next cycle and in how much they allowed the speed to change from cycle to cycle, but all of the policies used the minimum operating frequency as the lower bound on the speed. This is due to assuming that the system power was proportional to s^3, and consequently, that the minimum operating frequency results in minimum energy operation, as described in Chapter 2.

Using the CPU's minimum operating speed as the lower bound ignores the non-ideal behavior of the battery and the memory hierarchy, as well as the power consumed by the other subsystems, as shown by the results in Chapter 4. The following sections discuss the lower bound and a realistic method for finding it.

5.2 Determining the lower bound on speed

The lower bound can be determined either analytically or empirically, based upon the complexity of the functions describing the battery, power, and performance of the system. The work ratio presented in the previous chapter is an example of determining the lower bound analytically. As explained in Chapter 1, the computations per discharge is given by

$$Computations\ \ per\ \ Discharge(x) = \frac{BatteryCapacity(SystemPower(x))}{SystemPower(x)} \times Performance(x)$$

In general, the functions used to represent the battery capacity, system power, and performance may make it difficult to solve for the maximum of computations per discharge. For example, if the work per discharge is calculated using Peukert's formula for the battery capacity, assuming the system power as a function of frequency is $a_3f^3 + a_2f^2 + a_1f + a_0$, and assuming that the memory bandwidth is fixed, finding the maximum analytically involves solving a fourth-order polynomial equation. The overhead of finding the solution on-line may overshadow the savings that might be had.

But to gain some understanding of the lower bound on the range of speeds, it is useful to derive the computations per discharge for simple, common cases. Table 5.1 presents a list of common functions for each of the factors of the computations per discharge. In the

Table 5.1: Taxonomy of work ratio functions

Battery	System power versus frequency	Performance versus frequency
Ideal: $Q = k'$	Proportional: $P = af$	Ideal: $S = jf$
Non-ideal: $Q = k'/P^{\alpha}$	Linear: $P = a_1 f + a_0$	Memory Bottleneck: $S = j[1/(c + f_1(1-c)/f)]$
	Cubic Proportional: $P = af^3$	
	Cubic: $P = a_3 f^3 + a_2 f^2 + a_1 f + a_0$	

table, Q is the battery capacity, P is system power, f is frequency, S is performance, c is the fraction of memory accesses that access main memory, and a, a_x, f_1, j and k' are constants. Systems can be categorized with a notation of *Battery/SystemPower/Performance*. Thus the work ratio defined in the last chapter covers the *Non-ideal/Linear/Ideal* case, whereas the previous work in CPU speed-setting was the *Ideal/CubicProportional/Ideal* case. The most general case, *Non-ideal/Cubic/Bottleneck,* is a superset of the other cases: By setting α, c, a_x to 0 as appropriate, the general case becomes one of the other cases. However, as stated above, finding the maximum computations per discharge for the general case involves solving a fourth order polynomial equation.

While most of the entries in Table 5.1 are straightforward, *Memory Bottleneck* requires some explanation. The equation for *Memory Bottleneck* assumes that memory bandwidth is fixed and uses Amdahl's Law for the performance speed-up based upon the fraction of memory references that access main memory, c.

Table 5.2 shows the computations per discharge and the lower bound on speed for several common cases. In the "Lower bound on speed" column, f_{max} is the CPU's maximum

Table 5.2: Lower bounds for common cases

Case	Computations per discharge	Lower bound on speed
Ideal/Proportional/Ideal	$k'j/a$	None
Ideal/Linear/Ideal	$k' jf/(a_1 f + a_0)$	f_{max}
Ideal/CubicProportional/Ideal	$k' j/a_3 f^2$	0
Ideal/Cubic/Ideal	$k' jf/(a_3 f^3 + a_2 f^2 + a_1 f + a_0)$	$-2a_3 f^3 - a_2 f^2 + a_0 = 0$
Non-Ideal/Linear/Ideal	$k'jf/[(a_1 f + a_0)^{1+\alpha}]$	$(a_1 f+a_0)^{1+\alpha} - (1+\alpha)a_1 f = 0$
Ideal/Linear/Bottleneck	$k'j/[(a_1 f + a_0)(c + f_1(1-c)/f)]$	$[a_0 f_1(1-c)/a_1 c]^{0.5}$
Non-Ideal/Cubic/Bottleneck	$\dfrac{k'j}{[(a_3 f^3 + a_2 f^2 + a_1 f + a_0)^{1+\alpha}(c + f_1(1-c)/f)]}$	(fourth order equation)[a]

a. $-3(1+\alpha)a_3 cf^4 + (a_3-(1+\alpha)(2a_2 c+3a_3(1-c)f_1))f^3 + (a_2-(1+\alpha)(a_1 c + 2a_2 f_1(1-c)))f^2 + (a_1- a_1 f_1(1+\alpha)(1-c))f +a_0 = 0$
Solving this involves first solving a cubic equation and then a quadratic equation based upon the solution of the cubic [12].

operating frequency. It is interesting to note that only in the *Ideal/CubicProportional/Ideal* case is the lower bound 0. For most of the other cases, the lower bound is greater than 0 and determined by the system parameters rather than the minimum CPU operating frequency. For some combinations of parameters, it is possible that the lower bound is sufficiently close to f_{max} that there is little to be gained by having a speed-setting policy. The table enables mobile computer designers to check the parameters of their system, find what category their system falls into, and then by solving the appropriate equation decide whether a speed-setting policy would be useful. If so, then the policy requires a method to find the lower bound for the policy.

5.3 Interactions of the three factors

Finding a general solution to the lower bound is involved for all but the simplest cases. However, it is possible to make some generalizations even for the general case:

1. Non-ideal battery behavior decreases the lower bound.

2. Non-ideal performance behavior decreases the lower bound.
3. Static terms (a_0 in above tables) in the power function increases the lower bound.

It is assumed that for real systems, the coefficients of the system power function a_3, a_2, a_1, and a_0 are positive, that battery capacity is a decreasing function of power, and that performance is at most linearly increasing, i.e. it is bounded above by a linear function.

The proof for (1) is as follows: Let a = Performance(f)/Power(f) and b = Capacity(f). Assume $a > 0$, $b > 0$, and b is decreasing, i.e. $db/df \leq 0$. If a has only one maximum at f_{opt} for $0 < f < f_{max}$, then $da/df > 0$ for all $f < f_{opt}$, and $da/df < 0$ for all $f > f_{opt}$. Since

$$d(ab)/df = da/df \cdot b + db/df \cdot a,$$

then for $f = f_{opt}$,

$$d(ab)/df = 0 \cdot b + a \cdot (\text{negative number or } 0) = \text{negative number or } 0.$$

For $f > f_{opt}$,

$$d(ab)/df = (\text{negative number}) \cdot b + a \cdot (\text{negative number or } 0) = \text{negative number}.$$

For $f < f_{opt}$,

$$d(ab)/df = (\text{positive number}) \cdot b + a \cdot (\text{negative number or } 0) = \text{positive, negative, or } 0.$$

Then if there is a maximum for the product $a \cdot b$, it occurs for some $f_{new} \leq f_{opt}$. The range for the speeds used by a speed-setting policy is from f_{max} to f_{new}, and since $f_{new} \leq f_{opt}$ then this range is greater than or equal to the range f_{max} to f_{opt}. Thus non-ideal battery behavior decreases the lower bound.

A similar proof holds for (2), if it is assumed that performance as a function of frequency is not concave upwards. The range of speeds when performance is non-ideal is always greater than or equal to the range of speeds when performance is ideal. Thus non-ideal performance decreases the lower bound.

The proof for (3) is as follows: Let a = Capacity(f) · Performance(f). Let b = Power(f) - (static power). Let c = (static power). Then the computations per discharge is equal to

$$a/[b + c],$$

and the derivative of this is

$$[(b + c) \cdot da/df - db/df \cdot a]/[b + c]^2.$$

Let f_{opt} be the frequency that maximizes the computations per discharge. Then the numerator of the derivative,

$$[(b + c) \cdot da/df - db/df \cdot a],$$

is 0 for $f = f_{opt}$, is positive for $f < f_{opt}$, and is negative for $f > f_{opt}$. Now let c increase. Then the numerator becomes positive for $f = f_{opt}$ and $f < f_{opt}$, and positive, 0, or negative for $f > f_{opt}$, since a, b, and db/df are positive for real systems. Thus if there is a maximum when c is increased, it occurs for $f > f_{opt}$. And so, for the system power as a function of frequency, increasing the static term (a_0) increases the lower bound.

Since the three factors do not affect the lower bound in the same manner, it is necessary to calculate their relative effects in order to set the lower bound.

5.4 A general purpose method for finding the lower bound

There are at least three apparent on-line methods for a CPU speed-setting policy to find the lower bound. The first is to solve the fourth order equation in general and in each window of time, substitute the current values for the parameters of each of the functions. The solution would also have to checked to ensure that it is truly a maximum and not just an inflection point (a point where the slope is 0 but which is neither a minimum nor a maximum). The second method is to solve the fourth order equation iteratively, using Newton's method, for example. Again, the solution would have to be checked to ensure that it is truly a maximum. The final method is to use brute force, calculating the computations per discharge for each possible frequency and then choosing the frequency which gives the maximum value.

Brute force seems to be a more practical method for finding the lower bound. If the CPU utilizes a discrete set of frequencies as the StrongARM SA-1100 does, then the computations per discharge can be calculated for each frequency. If instead the CPU allows the frequency to be set to any value in a range, the computations per discharge can be calculated at equidistant frequencies across the range.

The brute force method for setting the lower bound, shown in pseudocode in Figure 5.2, is as follows. At the beginning of each window of time, the policy finds the current state of each device, looks up the power for each device in its respective state at each frequency and calculates the total power at each frequency. This total power is then used to calculate the capacity of the battery at each frequency. Next, the policy predicts the number of accesses to main memory for the next window of time and, from that, the performance at each frequency. Using the capacity, system power, and performance at each frequency, the policy calculates the computations per discharge for each frequency. The policy's lower bound on speed for the window is the greater of either the optimal frequency or the CPU's minimum operating frequency.

One of the elements of the *LowerBound* function is a table containing the power at each frequency of each device in the system for each of the device's power management states. This table is used to calculate the total power of the system. The maximum number of entries in this table is $d \times t \times n$, where d is the number of devices, t is the number of power

```
LowerBound()

for each CPU_frequency {
    total_power[CPU_frequency] = 0
    for each device {
        total_power[CPU_frequency] =
            total_power[CPU_frequency] + device_power[current_device_state]
    }
    capacity[CPU_frequency] = capacity_function(total_power[CPU_frequency])
    main_mem_accesses = predict_mem_access()
    performance[CPU_frequency] = performance_function(main_mem_accesses)
    computatations_per_discharge[CPU_frequency] = capacity[CPU_frequency] ×
        performance[CPU_frequency]/total_power[CPU_frequency]
}
max_value = 0
for each CPU_frequency {
    if computations_per_discharge[CPU_frequency] > max_value {
        optimal_frequency = CPU_frequency
        max_value = computations_per_discharge[CPU_frequency]
    }
}
return maximum(optimal_frequency, minimum_operating_frequency)
```

Figure 5.2 Pseudo-code for finding lower bound using brute force method.

management states per device, and *n* is the number of possible frequencies. Each device could provide such a table as part of its device driver code.

The battery capacity function could be implemented either as an analytical formula or as a look-up table. This function would vary from battery to battery. Essentially, the battery would deal with the operating system like any other device, with a device driver that provides this function. Alternatively, the battery device driver could provide simply the battery's make and model number, and the policy would then choose an appropriate capacity function for that battery from a list of possible capacity functions.

Finally, the performance function is the most difficult element of *LowerBound*. Predicting performance over some window of time is likely a topic for a dissertation by itself. However, given that the window of time is on the order of tens of milliseconds or more, the prediction is much less difficult than if the window of time were on the order of the clock cycles, as any large variations will be averaged over a longer time. The main assumption is that performance will scale with frequency except for main memory accesses. Thus predicting performance is a problem of predicting main memory accesses.

The *LowerBound* function corrects several shortcomings of the speed-setting policies described in the previous work. First, it accounts for lower order terms in the function of system power versus frequency. Second, it allows for a change in the system power based upon the power management state of each of the subsystems. Third, it considers the loss of battery capacity with increasing power. Finally, it accounts for non-ideal performance speed-up due to limitations of the memory hierarchy.

Figure 5.3 shows pseudo-code for the updated version of the speed-setting policy *Past*. The only change is to set the minimum CPU speed to the value returned by *LowerBound*. The other policies described in the previous work can be modified in the same way.

While it is not possible at this time to test *LowerBound* with an actual system, some confidence in the new lower bounds can be gained by re-visiting the simulations of Weiser et al. Table 5.3 shows the results of the Weiser et al. simulator using the updated policy of Figure 5.3. The system power function was assumed to be $0.9s^3 + 0.1$ in column (a), $0.75s^3 + 0.25$ in column (b), and $0.5s^3 + 0.5$ in column (c), i.e. the static term is 10%, 20%, and 50% of the total power at full speed, with s being the normalized speed. The simulator settings chosen for comparison were those which were determined to be the best by Weiser et al., a minimum value of $s = 0.44$ and a window size of 20 ms. Most of the traces were 10 to 60 seconds long, although one (kestrel) was nearly 10 hours long. All results are normalized to what would have been achieved had the CPU run at full speed for the whole trace.

Since the static term in the power function of column (a) is relatively small (10%), the results for both the updated and original policy are nearly the same, as expected. Because

```
Past()

min_cpu_speed = LowerBound()

new_cpu_speed = old_cpu_speed
if excess_cycles > idle_cycles {
    new_cpu_speed = 1.0
    return
}
run_percent =
    run_cycles/(run_cycles+idle_cycles)
if run_percent > RUN_HIGH
    new_cpu_speed =
        new_cpu_speed + Delta
if run_percent < RUN_LOW
    new_cpu_speed =
        new_cpu_speed - Delta
if new_cpu_speed > 1.0
    new_cpu_speed = 1.0
if new_cpu_speed < min_cpu_speed
    new_cpu_speed = min_cpu_speed
return
```

Figure 5.3 Pseudo-code for speed-setting policy *Past* updated to use the *LowerBound* function

the lower bound on speed for this case was less than the minimum determined to be best by Weiser et al., *LowerBound* performs slightly worse for some traces than the original policy. As Weiser et al. reported, running too slowly can sometimes lead to larger energy consumption because cycles that cannot be completed in a window have to be made up at a faster speed in the next window. If *LowerBound* had used the minimum operating frequency suggested by Weiser et al., the results for this column would have been identical for both it and the original policy. Note that the best possible increase for column (a) is 2.54. That neither policy achieves this is due to the performance constraints of the trace. Achieving the best possible increase occurs only when there is sufficient idle time in each window for the CPU to always run at the optimal frequency. If there is not enough idle

Table 5.3 Comparison of normalized computations per discharge for updated policy and original Weiser policy

Trace name	(a): Power = $0.9s^3 + 0.1$ Updated / Weiser	(b): Power = $0.75s^3 + 0.25$ Updated / Weiser	(c): Power = $0.5s^3 + 0.5$ Updated / Weiser
heur1	1.03 / 1.03	1.03 / 1.03	1.02 / 1.02
fm1	1.23 / 1.23	1.15 / 1.13	1.03 / 0.99
idle1	1.54 / 1.56	1.27 / 1.23	1.04 / 0.92
em3	1.30 /1.30	1.17 / 1.15	1.02 / 0.96
mx2	1.81 / 1.79	1.32 / 1.28	1.05 / 0.87
emacs2	1.74 / 1.75	1.31 / 1.27	1.05 / 0.87
mx3	1.23 / 1.23	1.14 / 1.13	1.03 / 0.99
emacs1	1.71 / 1.71	1.33 / 1.28	1.06 / 0.89
mx1	1.73 / 1.74	1.32 / 1.29	1.06 / 0.90
fm2	1.31 / 1.32	1.20 / 1.17	1.04 / 0.98
fm3	1.06 / 1.06	1.04 / 1.04	1.02 / 1.00
kestrel	1.81 / 1.82	1.34 / 1.29	1.05 / 0.87

time, then the CPU runs faster than the optimal frequency, which lowers the normalized computations per discharge.

In column (b), the static term is slightly larger (25%), and here the updated policy begins to consistently outperform the original. For column (b), the best possible increase is 1.47.

In column (c), the updated policy shows its merit, as it is able to increase the computations per discharge for every trace while the original policy causes them to decrease for every trace. The best possible increase for column (c) is 1.06. The updated policy performed nearly this well for several traces, including the longest trace, kestrel.

The results show that the updated policy performs as well as the original policy when the system power is nearly ideal and outperforms it as the system power becomes less ideal.

Having the lower bound determined by the system characteristics is better than having it determined by the CPU's minimum operating frequency. Using the minimum operating frequency, the computations per discharge decreases in some cases, since the lower order term in the power function cause the energy consumption to increase if the speed is set too low.

These results provide only a limited validation of the realistic speed-setting policy. The simulation and the accompanying traces had no information about the states of subsystems in each window or even which subsystems were accessed in each window. Thus these results show only that setting the lower bound on speed according to the lower order power terms prevents a policy from causing the computations per discharge to decrease. This is in itself a large step toward taking a system level approach to the speed-setting problem, as the policy no longer focuses on only the CPU power. To be fully validated, however, the *LowerBound* function must be implemented on a mobile system, which in turn requires that the system have several features.

5.5 System features for implementing a speed-setting policy

Because a realistic CPU speed-setting policy must consider aspects of the system other than simply idle time, the system should have features that account for battery behavior, total system power, and actual application performance as the CPU frequency is changed.

For the battery behavior, there are "gas-gauge" IC's which keep track of the charge in and out of the battery, the current load, and the battery family. The operating system can then treat the battery as any other device in the system with a device driver for each type of battery. The operating system can probe the battery to determine its type and set the battery capacity function parameters accordingly.

For performance, the CPU should have performance-monitoring registers such as those found in high-end CPU's like the Alpha and the Pentium. CPU's intended for mobile use tend to not have these registers, but as was shown by the Itsy performance measurements in Chapter 4, such registers could be useful in mobile computing. The operating system could then monitor the current performance behavior when making CPU speed-setting decisions.

Finally, for the total system power, the “gas-gauge” IC’s would again be useful as they can report the current load, augmenting the look-up table in the operating system giving the power for each subsystem and each of its states. The values in the table could initially be the manufacturer’s typical values for the devices. Then the “gas-gauge” IC could be used by the policy to correct the total power calculation for common combinations of device states.

5.6 Summary

This chapter first discussed the goals of a CPU speed-setting policy. It then described a notation for a system’s battery capacity, power, and performance, and used the notation to derive lower bounds on useful CPU speeds for simple cases. Finally, it outlined the elements of a realistic, general purpose method for finding the lower bound on CPU speed, and applied it to an existing CPU speed-setting policy.

A policy for CPU speed-setting should take a system level view, taking into account the battery behavior, the actual performance of the application code, and the total system power. The technology to account for all three exists, but has not been integrated. The goals of the policy should be to first give the same apparent performance as if the CPU were running at maximum speed at all times, and second, to maximize the work completed in a discharge. These two goals provide upper and lower bounds on the speeds that should be used by a policy. The upper bound is maximum CPU frequency. The lower bound is the speed which maximizes the computations per discharge. Running more slowly than this means decreasing both performance and the work completed in a discharge, both undesirable. A brute force calculation of the computations per discharge at each frequency, as outlined in this chapter, should provide an adequate means for finding the lower bound. The lower bound calculated in this manner is realistic as it accounts for non-ideal battery, power, and performance behavior, and is general purpose because it can be applied to any implementation of a policy.

Chapter 6

Conclusion

This chapter summarizes the thesis, lists the contributions to the areas of low power and mobile computing, and describes future extensions and other applications.

6.1 Summary

This work has combined models of power source behavior with models of system performance to better understand the trade-offs between performance and power in low power computing. Computations per discharge should be used to evaluate such trade-offs, as it captures important aspects of the battery capacity, the total system power, and the actual performance. Ignoring any one of these three may lead to the trade-off being made incorrectly.

Two non-ideal battery properties with the potential to affect power-performance trade-offs are loss of capacity and recovery. Chapter 3 showed that recovery is not a problem for the loads and batteries typical of mobile computing. But the loss of capacity means that peak power rather than average power determines battery capacity. Consequently, reducing the peak power of a mobile system will increase the battery life by more than reducing the idle power, even if both reductions result in the same average power. An important method of reducing peak power is trading power for performance, particularly setting the CPU speed.

Previous work in CPU speed-setting attempted to minimize the energy per operation of the CPU. It assumed that the voltage of the CPU must scale with the speed and that the power varies as s^3, where s is the factor by which both the frequency and voltage are scaled. The previous work further assumed that the performance also scaled with speed. These assumptions do not account for the battery behavior, the total system power, or the actual performance. For real systems, a policy should maximize the computations per battery life rather than minimizing the energy per operation. It should assume that the system power will not be proportional to s^3 but will have lower order terms due to the other subsystems. It should account for the performance not scaling with CPU frequency when the memory hierarchy is a bottleneck. That these factors should be considered was confirmed by results from battery discharge experiments with mobile systems, which were up to nearly a factor of four less than expected because of non-ideal battery and performance behavior.

A practical method for determining the lower bound on speeds for a CPU speed-setting policy was described. This method accounts for battery behavior, total system power, and actual performance, as well as changes in those aspects of the system due to power management states of the subsystems. The lower bound can be used with any speed-setting policy, as all of them share the goals of first meeting user performance expectations and then maximizing the computations per battery life. Existing technology can be used to furnish the information that this method for the lower bound requires.

6.2 Summary of contributions

The research described in this thesis contributes to the areas of low power and mobile computing in the following ways:

- Regions of ideal/non-ideal battery behavior were delineated, allowing system designers to understand when measuring only average power is adequate, and when dynamic power must also be considered.
- Typical bounds on the approximate effect of battery capacity loss were given.
- Continuous discharge behavior was shown to be useful for estimating intermittent discharge behavior.

- Reducing idle power was found to have less of an effect than reducing active power when the active power lies in the non-ideal range of battery behavior.
- It is the only system level power research to use battery discharge experiments to verify predictions.
- The memory hierarchy was shown to be an important consideration when setting the CPU speed.
- A realistic policy for dynamically setting the CPU speed was described.

6.3 Future work

The work reported in this thesis can be extended in several ways as well as applied to other areas of research.

One extension would be to use the pieces of the phenomenological models described in Chapter 3 to create a new phenomenological model, one applicable to a wider variety of batteries. Doyle's model requires nearly 50 parameters to describe the battery, many of which deal with details of the battery's construction and manufacture and are unavailable to the everyday user. A phenomenological model would allow the end user to determine parameters from a few measurements of the cell to be modeled or from published manufacturer data. A phenomenological model could be used by the *LowerBound* function or in formulating analytical solutions to other types of power-performance trade-offs. An example of the latter would be augmenting Zorzi's approach to network protocols [76] with a phenomenological battery model appropriate for a Markov chain.

The most pressing extension is to verify the *LowerBound* function described in Chapter 5, using either simulation or a real system. Simulation would involve gathering traces from actual mobile systems, building a simulator similar to that of Weiser et al. [72] but augmented to account for the behavior of the memory hierarchy and the power management state of each subsystems, and creating a power profile to be input either into a first-principles model such as Doyle's (slow) or a phenomenological model such as above (fast). Verification with a real system requires a CPU whose speed can be changed by software, and some method of monitoring battery capacity, total power, and system performance.

A possible extension to the policy itself is that early in a discharge the battery should be treated as ideal, but later in the discharge, it should be treated as non-ideal. Then the policy would change modes after a certain point in the discharge. Thus it may be that a better approach is to minimize energy per operation early in the discharge and maximize computations per discharge late in the discharge. Because the peak power that can be handled by the battery decreases with the depth of discharge, it is possible that events early in the discharge begin to look to the battery like their average value, i.e., when the battery is 80% discharged, it does not matter how it came to be 80% discharged, just that it is 80% discharged. Thus the proper approach is to minimize the energy per operation early in the discharge, and only be concerned with non-ideal battery properties after the battery has reached a depth of discharge where the peak power of the system would cause the battery to reach its end-of-discharge voltage.

Another extension would be to investigate I/O power-performance trade-offs. I/O bandwidth may have an impact similar to limited memory bandwidth.

The research can be applied to other areas as well. When measuring the energy usage of applications [23][54], the peak power must be considered because of the findings of Chapter 3. Resource scheduling, especially prefetching from hard drives and across wireless networks, is another area where the findings of Chapter 3 can be applied. Both disks and wireless devices can generate large peak powers. The way to reduce the overall peaks is to make the disk and network generate their respective peaks when the rest of the system is idle.

Another application for this research is autonomous mobile robots. Battery-powered mobile robots must decide between how fast they travel and how far they can travel, a trade-off similar to that of performance versus computations per discharge. Suppose a mobile robot has collected samples and must return them. The robot should travel at a speed which maximizes the distance it can travel. Either it will reach the destination or it will come as close as possible to reaching it (which increases the chance it can be retrieved by another robot). Slippage of the wheels is similar to the loss of performance due to the memory hierarchy. An added problem is dealing with turning. The path the robot takes is not necessarily (nor likely to be) a straight line, so rate of turning and the friction involved must also be accounted for.

Finally, another area would be to study other applications where performance versus CPU frequency is an issue. This work looked at CPU speed-setting only for power-performance trade-offs. Examining performance versus CPU frequency may also be useful for embedded systems, as suggested by [27], which calls for compilers for embedded systems to have knowledge of the performance versus frequency behavior of the processor.

Appendix A

Derivation of the work ratio

In the previous work described in Chapter 2, it was assumed that the power consumed by the system was directly proportional to the CPU clock frequency. A more general assumption is that the current drawn by a system has two components, S and CV^2f, where S is the component of the current that is independent of the CPU frequency f, and CV^2f, the CMOS dynamic power, is the component that depends on the CPU frequency,

$$I = S + CV^2f$$

The power of the system, P, is then

$$P = V_0(S + CV^2f)$$

where V_0 is the voltage of the battery. This more general relationship allows one to represent the full range of possibilities, from the system of the earlier example, where the power of the system is directly proportional to the CPU clock frequency (S=0), to systems where the system power does not depend on the CPU frequency (CV^2f=0). The charge capacity of a battery, Q, is related to its discharge current I by Peukert's formula,

$$Q = k/I^{\alpha},$$

The life of the battery, then, is

$$L = V_0Q/P = V_0(k/I^{\alpha})/[V_0(S + CV^2f)] =$$
$$k/(S + CV^2f)I^{\alpha} = k/(S + CV^2f)^{1+\alpha}$$

For this example, computation will be measured by the number of iterations of a loop that can be performed. Let j be the number of iterations of the loop executed per unit time. Then the total number of iterations the system can complete during a battery life is

$$W = jL = jk/(S + CV^2f)^{1+\alpha}$$

If the CPU frequency is changed by a factor n, $n > 0$, then the CPU frequency dependent portion of the power changes from CV^2f to CV^2fn, and, ideally, the number of iterations executed per unit time from j to jn. Then the total number of iterations the system can complete is

$$W_n = njL = njk/(S + CV^2fn)^{1+\alpha}$$

Taking the ratio W_n/W gives

$$W_n/W = [njk/(S + CV^2fn)^{1+\alpha}]/[jk/(S + CV^2f)^{1+\alpha}] =$$

$$n\left(\frac{S + fCV^2}{S + fnCV^2}\right)^{1+\alpha}$$

Letting $\rho = S/(S + CV^2f)$, the percent of the system power that is independent of the CPU clock frequency at the initial frequency, then

$$W_n/W = n\left(\frac{S/\rho}{(S/\rho) + (n-1)(S/\rho)(1-\rho)}\right)^{1+\alpha}$$

$$= \quad n\left(\frac{1}{\rho + n(1-\rho)}\right)^{1+\alpha}$$

Appendix B

Comparison of the work ratio to existing metrics

Unlike previous metrics relating performance and power, the work ratio shows that in some situations there is a useful trade-off between performance and battery life. An occasionally used power metric for CPU's intended for portable applications is MIPS/Watt [7][51]. An analysis of units shows that MIPS/Watt is actually (millions of instructions/s)/(J/s) = millions of instructions/J. For a given CPU, increasing its clock frequency from f to nf increases its MIPS rating by a factor of n, assuming that other factors such as I/O do not become bottlenecks. Increasing the clock frequency also increases the power, but by less than a factor of n because power has a static component and a dynamic component: $S + CV^2f$. Because $f/(S + CV^2f) < nf/(S + CV^2nf)$ for $n > 1$, the CPU running at a higher frequency will always have a higher MIPS/W metric than it does when running at a lower frequency. MIPS/W, then, is monotonically increasing, and does not give any sense of diminishing returns.

Another metric in the literature power-delay, which for CPU's has the units W/Spec^2 [34]. Unlike MIPS/W, the power-delay metric has the property that "smaller is better". Again, increasing the clock frequency from f to nf increases its Spec rating by a factor of n, while the power increases by less than a factor of n. So a CPU running at a higher frequency will always have a lower W/Spec^2 rating than it does when running at a lower frequency. As with MIPS/W, W/Spec^2 does not give any sense of diminishing returns.

The work ratio gives a sense of diminishing returns. It does not always become better with increasing frequency. Taking the partial derivative with respect to the speedup factor,

$$\partial W_n/\partial W = [-n(1 + \alpha)(1 - \rho) + (\rho + n(1 - \rho))]/[(\rho + n(1 - \rho))]^{-(2+\alpha)}$$

Setting the numerator equal to 0 and solving for n, one finds that the maximum occurs at n $= \rho/[\alpha(1 - \rho)]$

So, unlike the previous metrics, the work ratio has a point where increasing the CPU frequency is no longer beneficial. By substituting the above value of n into the equation for ρ, one finds that the optimal point is where $\rho = \alpha/(\alpha+1)$. Whether most systems can operate at this point is a question for further investigation.

Appendix C

Brief review of the properties of batteries

"Remember that any high-energy, high-power battery is a potential bomb."
John Newman, [52]

This appendix is a brief review of the properties of batteries and common definitions associated with them. It begins with the basic operation of a battery and defines common terms. Following the definitions are descriptions of ideal and non-ideal battery properties. The differences between the two, especially the non-ideal transient behavior, are the basis for this work's assertion that the characteristics of the battery must be taken into account to properly balance power and performance in mobile systems. There follows a discussion of porous electrode theory, which underlies Doyle's model presented in Chapter 3.

C.1 Basic Operation

A battery is a set of one or more electrochemical cells, connected in either series or parallel. A cell consists of two electrodes, an anode and a cathode, separated by an electrolyte. During discharge, the positive electrode is referred to as the cathode and the negative electrode as the anode. When an external load is connected to the battery, electronic current flows through the load and ionic current flows through the electrolyte, as shown in Figure C.1. The charge of the ions and the direction of their flow is determined by the materials used for the electrodes and electrolyte. If the majority carrier in the electrolyte is positive, then the ions flow from the anode to the cathode. If the majority carrier is negative, then the ions flow from the cathode to the anode. There may be minority carrier flow as well.

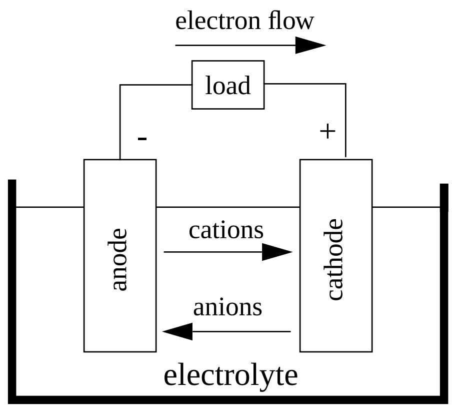


Figure C.1 Cell during discharge

After the battery is discharged, it is either discarded or recharged. Primary batteries can be discharged only once, while secondary batteries are rechargeable. To recharge a secondary battery, an external voltage source is connected to it as shown in Figure C.2. During the charge cycle, the positive electrode is referred to as the anode and the negative electrode as the cathode, because the flow of charge at the electrodes is the reverse of the flow during discharge. While the charging method is very important to the health of a battery, its properties during discharge, and the number of discharges it can give, this dissertation will deal only with the discharge cycle. Charging has been treated in detail elsewhere [28][66], and battery manufacturers typically specify the charge regimen in their data sheets. This dis-

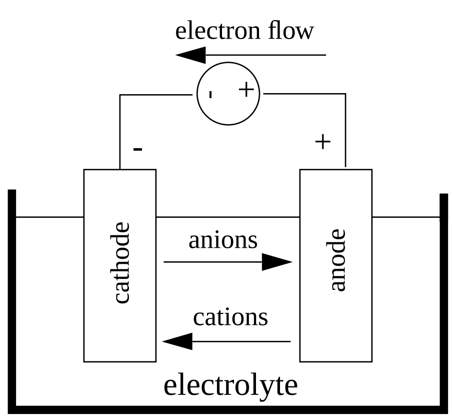


Figure C.2 Cell during charge

sertation examines ways to better utilize the battery's capacity during the discharge cycle, assuming that the charge method gives equal capacity at the start of the discharge.

C.2 Ideal Voltage and Capacity

The voltage and capacity of a battery are the properties of most immediate concern to mobile systems, followed by the battery's weight, volume, and safety. The theoretical voltage and capacity are determined by the materials used for the electrodes. Table C.1 lists the standard potentials and electrochemical equivalents of several common electrode materials. The theoretical potential is found by subtracting the standard reduction potential

Table C.1: Standard potentials and electrochemical equivalents of common electrode materials

Material	Standard reduction potential, V	Electrochemical equivalent, g/Ah
Anode materials		
H_2	0	0.037
Li	-3.01	0.259
Cd*	-0.81	2.10
Pb	-0.13	3.87
Cathode materials		
SO_2	1.36	2.38
MnO_2	1.23	3.24
NiOOH	0.49	3.42
PbO_2	1.69	4.45

*Using the reaction $Cd(OH)_2 + 2e \Leftrightarrow Cd + 2OH^-$

of the anode from that of the cathode. For example, the theoretical voltage of a NiCd battery is

$$0.49\ V\ [\mathrm{NiOOH}] - (-0.81\ V)\ [\mathrm{Cd}] = 1.30\ V$$

Further information on the standard potentials is available in Newman [52].

While the previous treatment of the theoretical voltage using standard potentials is usual, it is instructive to discuss the theoretical voltage in terms of the Fermi levels in light of the semiconductor background of the intended audience. Figure C.3 shows the relationship

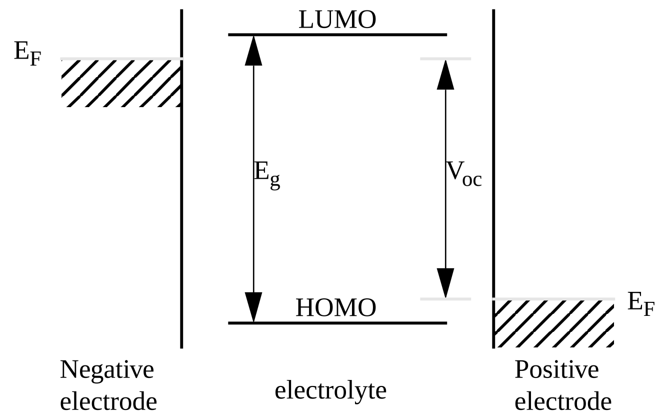


Figure C.3 Fermi levels of battery

between the open- circuit voltage and the Fermi levels of the electrodes [30]. The open-circuit voltage is equal to the magnitude of the difference between the Fermi levels of the electrodes.

Figure C.3 also shows how the choice of electrodes impacts the choice of electrolyte. In general, for the electrolyte to not react chemically with the electrodes, its lowest unoccupied molecular orbit (LUMO) must be above the Fermi level of the negative electrode, while its highest occupied molecular orbital (HOMO) must be below the Fermi level of the positive electrode. If a battery family has a large open-circuit voltage, then the choice of electrolyte narrows to those with a large energy gap E_g. For example, the open-circuit voltage of Li-ion batteries rules out aqueous electrolytes, which have an E_g of 1.23 eV.

While the voltage depends on the Fermi levels of the electrode materials, the capacity depends on their molecular weights. The capacity of a battery usually refers to its charge capacity, which is given in Amp-hours, abbreviated Ah. (1 Ah = 3600 Coulombs.) For example, a battery which can be discharged for 5 hours at 0.3 A has a capacity of 1.5 Ah. The charge capacity is commonly called the battery's "C rating." Capacity is also used to refer to a battery's energy capacity, typically given in Watt-hours. The energy capacity is the nominal voltage times the charge capacity. In this dissertation capacity will refer to energy capacity rather than charge capacity unless noted otherwise. Terms used to relate the capacity include specific energy, the energy capacity per unit weight, typically given in Wh/kg; energy density, the energy capacity per unit volume, Wh/L; specific power, the power delivered by the battery per unit weight, W/kg; and power density, the power delivered per unit volume, W/L. Energy density and power density are occasionally used to refer to the specific energy and specific power. To avoid confusion, this dissertation will not use the terms energy density or power density.

The theoretical charge capacity is calculated from the gram equivalent weights of the electrodes. A gram equivalent weight of a material is its atomic weight in grams divided by the number of electrons produced or consumed in its reaction. According to Faraday's Laws, one gram equivalent will produce 26.8 Ah of charge. Thus for the NiCd battery, using the values from Table C.1, the theoretical charge capacity is

$$3.42\ g/Ah\ [\mathrm{NiOOH}] + 2.10\ g/Ah\ [\mathrm{Cd}] = 5.52\ g/Ah,\ or\ 181\ Ah/kg$$

The theoretical specific energy of the NiCd battery family is then 1.30 V $\times$ 181 Ah/kg = 235 Wh/kg.

The theoretical voltage and capacity consider only the electrode materials and do not reflect contributions due to the electrolyte, cell housing, and other construction components. Consequently, the practical voltage and capacity are often considerably less than the theoretical values. In addition, the voltage and capacity vary with the load.

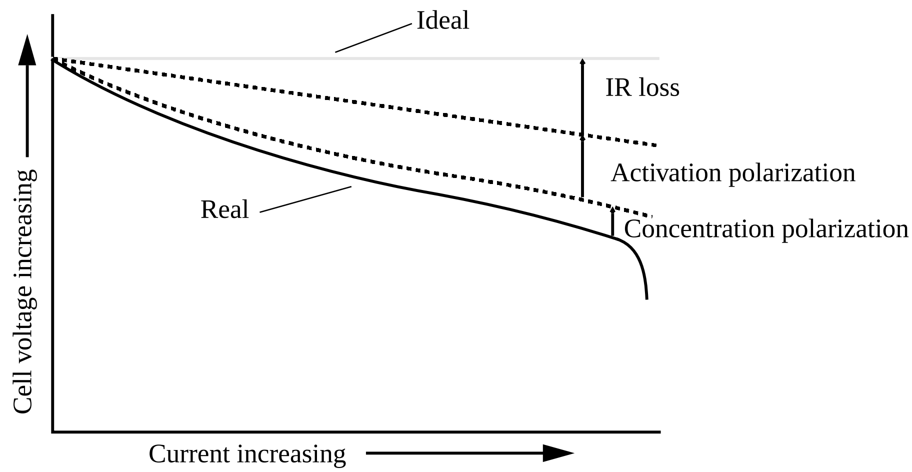


Figure C.4 Cell polarization vs. operating current, after [43], Figure 2.1

C.3 Non-ideal Voltage and Capacity

An ideal battery's charge capacity and voltage (and hence energy capacity) are constant for all values of the load. In practice, however, both the voltage and the charge capacity decrease as the load increases. Voltage losses are due to the activation polarization, concentration polarization, and ohmic polarization, shown in Figure C.4 [43]. Activation polarization or surface overpotential is the driving force of the chemical reaction at the electrode/electrolyte interface. Concentration polarization or concentration overpotential is the voltage drop due to differences in the concentration of the electrolyte at the reaction site and in the bulk. Ohmic polarization, or IR loss, is the resistive drop through the electrolyte and electrodes, and across the contacts between the electrodes and the current collectors. The magnitudes of these losses are determined in part by the battery family, and in part by the physical design of the battery. For example, the electrolyte should be conductive enough that ohmic losses are not large in the intended region of operation, while porous electrodes can be used to reduce activation polarization losses, decreasing current density by increasing the surface area of the reaction. Porous electrodes will be discussed further in Section C.4.

C.3.1 Expressions for concentration and activation polarization

While the ohmic polarization varies linearly with the discharge current, the concentration and activation overpotentials do not. A greatly simplified but still instructive expression for the concentration polarization η_c is

$$\eta_c = \frac{RT}{nF} ln\frac{c_b}{c_o} \quad (Volts)$$

where c_o is the electrolyte concentration at the electrode, c_b is the concentration in the bulk, R is the gas constant, F is Faraday's constant, T is the temperature, and n is the number of electrons involved in the reaction [43]. As the battery discharges, the electrolyte concentration near the electrode decreases, causing the concentration polarization to increase.

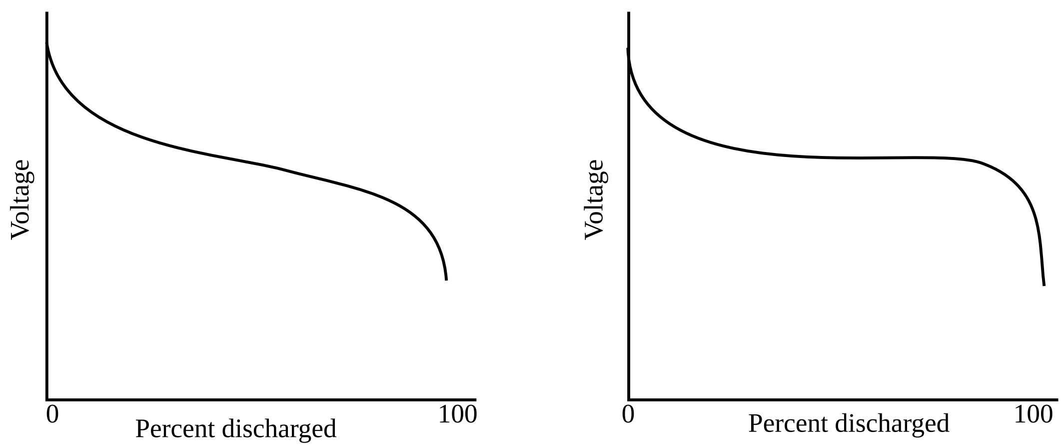
For the activation polarization, a simplified expression is

$$\eta_s = \frac{RT}{\alpha F} ln\frac{i}{i_0} \quad (Volts)$$

where i is the current density flowing between the electrodes, i_0 is the exchange current density and α is the activation coefficient [52]. The exchange current density i_0 is a measure of the rate of the reaction under equilibrium conditions, when no net current is flowing. This expression for the activation polarization shows that the magnitude of η_s increases as the current density i is increased.

C.3.2 Shape of the discharge voltage profile

The shape of the discharge voltage profile varies from battery family to battery family. Some families have a sloped voltage profile, while others have a nearly flat discharge voltage profile, as shown in Figure C.5 a and b. Note that both profiles have steep slopes at the



a. Sloped discharge profile   b. Flat discharge profile

Figure C.5 Typical discharge voltage profiles

beginning and end of discharge. The concentration and activation overpotential expressions given in Section C.3.1 help explain the steep slopes. At the beginning of the discharge, reactants are used up at the reaction sites, setting up concentration gradients that cause reactants to diffuse from the bulk of the electrodes and separator. Until the gradients reach a steady state, the concentration overpotentials of the electrodes change. At the end of discharge, reactants are exhausted in some areas of the electrode before they are exhausted elsewhere, resulting in increased current density in the areas where the reactants are still available. The increased current density increases the activation polarization. The increased current density also creates larger concentration gradients, so the concentration potential increases as well.

There are several factors which determine the slope of the profile in the middle of the discharge:

- Changes in resistance with discharge. If the conductivity of the reactants and products are much different, then as the discharge reaction occurs the resistance of the cell will change. The change in resistance also depends on the localization of the reaction site. If the reaction occurs at nearly the same rate throughout the electrode, the resistance change during discharge will be less than if reaction occurs in a localized area that moves through the electrode during the discharge.

- Changes in the standard potential. The standard potential of some electrodes varies as the electrode composition changes with the reaction. This is the case with many Li-ion batteries, for example.

The shape of the discharge profile impacts the design of the power supply and the method of determining state of discharge. If the profile is flat, then it is possible to use a power supply with a narrow input voltage range. If the profile is sloped, then the power supply must be able to operate over a wider range of input voltages. As for determining the state of discharge, if the profile is sloped, then the battery's voltage is a good indicator of the remaining capacity. If the profile is flat, however, then there may be only a few tens of millivolts change in the voltage as the battery discharges from 90% capacity to 10%.

A battery is discharged when its voltage drops below a specified value, called the cutoff voltage. Factors that determine the cutoff voltage include the battery family, the number of cells which make up the battery, and the rate of the load [28]. Discharging a battery below its cutoff voltage may damage it, rendering it immediately useless or reducing the number of cycles it can be used. The active materials in the battery may not be fully utilized when the battery voltage reaches the cutoff value, due to the voltage drop from activation, concentration, and ohmic polarization. If the active material is not fully utilized, the capacity of the battery is reduced.

C.3.3 Loss of capacity

The power lost due to activation, concentration, and ohmic polarization cannot be recovered [43]. It is consumed like any other power, given off as heat. Figure C.6 shows a Ragone plot, a log-log plot of specific energy vs. specific power [43], for the Canon BP-911 Li-ion battery, generated by the method described by Doyle et al. [19]. The Ragone plot uses specific energy vs. specific power so that one plot can describe a family of batteries using the same materials but with different capacities, i.e. a battery of a particular family with a capacity of 1 Ah would be described by the same Ragone plot as a battery of the same family with a capacity of 2 Ah. For the BP-911, the capacity at the highest rate shown is nearly 40% less than it is at the lowest rate. For other batteries, the amount of lost capacity, the slopes of the curve at the low and high ends, and the point at which the capacity begins to decrease will differ. But the general trends shown in the Ragone plot of

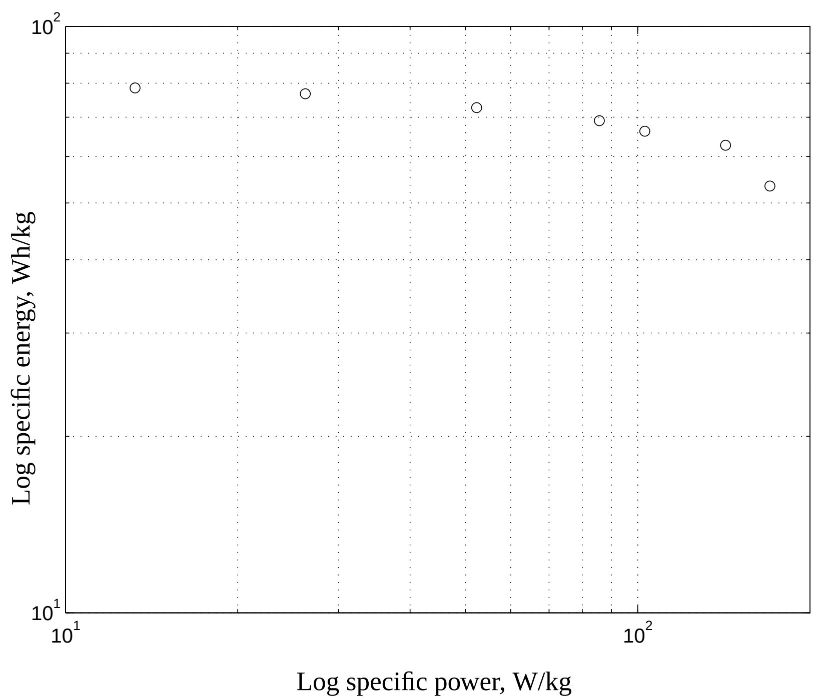


Figure C.6 Ragone plot for the Canon BP-911

the BP-911 are typical of most batteries. Only at low rates is the energy capacity constant. As the rate increases, the capacity decreases.

While the losses due to activation, concentration, and ohmic polarization cannot be recovered, they do not entirely explain the lost capacity at higher power shown in the Ragone plot of Figure C.6. The capacity is lost at higher rates because the cutoff voltage is reached before the active materials of the battery are exhausted. The “lost” capacity at a given discharge rate is available at a lower rate, or at the same rate if the load is removed for some time.

C.3.4 Transient behavior

The properties discussed so far have been steady state properties. The discharge profiles given have assumed constant loads throughout the discharge. The mobile computers this dissertation is concerned with, however, have discharges that are anything but constant.

Figure C.7 shows the voltage profiles of a constant discharge and an intermittent discharge. For the sake of illustration, the off time of the intermittent discharge is not shown.

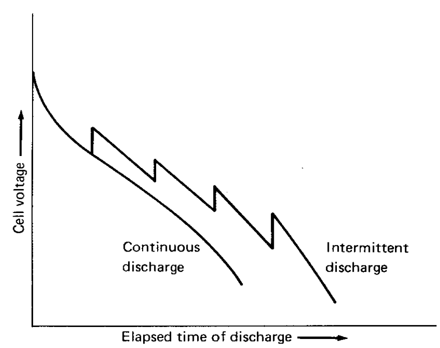


Figure C.7 Battery recovery (after [43])

Both the constant discharge and the intermittent discharge, while on, have the same discharge rate. Consequently, the time of discharge is proportional to the delivered capacity. The intermittent discharge delivers greater capacity before the cutoff voltage is reached than the constant discharge. The difference in capacity will depend on the rate of the discharge, the off time of the intermittent discharge, and the physical design of the cell.

An intuitive explanation of why the intermittent discharge delivers greater capacity depends on the concentration overpotential. During the off time, concentration gradients between the bulk of the electrolyte and that near the surface of the electrode decrease due to diffusion of the ions. If the off time is sufficiently long that the gradient is greatly or entirely reduced, then when the load turns on, the concentration overpotential will be smaller than it was when the load was turned off. This is the cause of the voltage peaks in Figure C.7. As the load stays on, the concentration overpotential will return to the steady state value. However, due to the diffusion that occurs during the off time, more of the reactants will be used than in the continuous discharge. Hence the greater capacity.

This simplified explanation illustrates why the difference in capacity between high rate intermittent and constant discharges may be greater than the difference in capacity for low rate intermittent and constant discharges. The concentration gradients set up by the high rate discharges will be greater, so that more diffusion will occur during the off time than for a low rate discharge. In addition, the continuous high rate discharge will use less of the total capacity of the battery than a low rate continuous discharge, so there is a larger amount of capacity available to the intermittent high rate discharge than to the intermittent low rate discharge, as shown in Figure C.8.

C.4 Porous Electrodes

Accurate analysis of battery behavior is not possible without taking into account the effects of porous electrodes. The following is a basic review of porous electrode theory in order to prepare the reader for the details of the battery models presented in the next chapter. This section is drawn mainly from Chapter 22 of Newman [52], which contains an in-depth discussion of the topic.

Figure C.9 shows the cross-section of a cell with a porous electrode. One or both electrodes of most cells are porous. Porous electrodes have several advantages over plane electrodes, the most important of which for battery applications are:

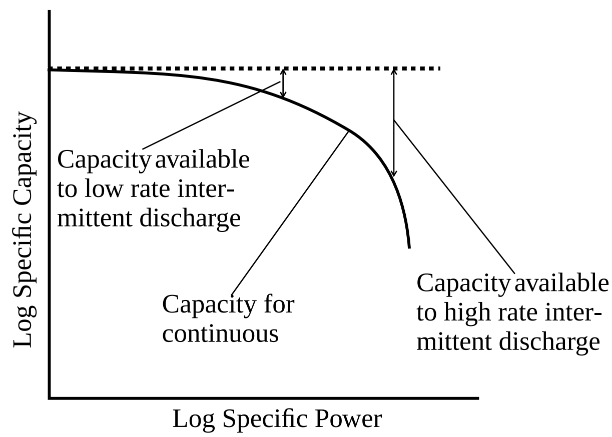


Figure C.8 Capacity available to an intermittent discharge

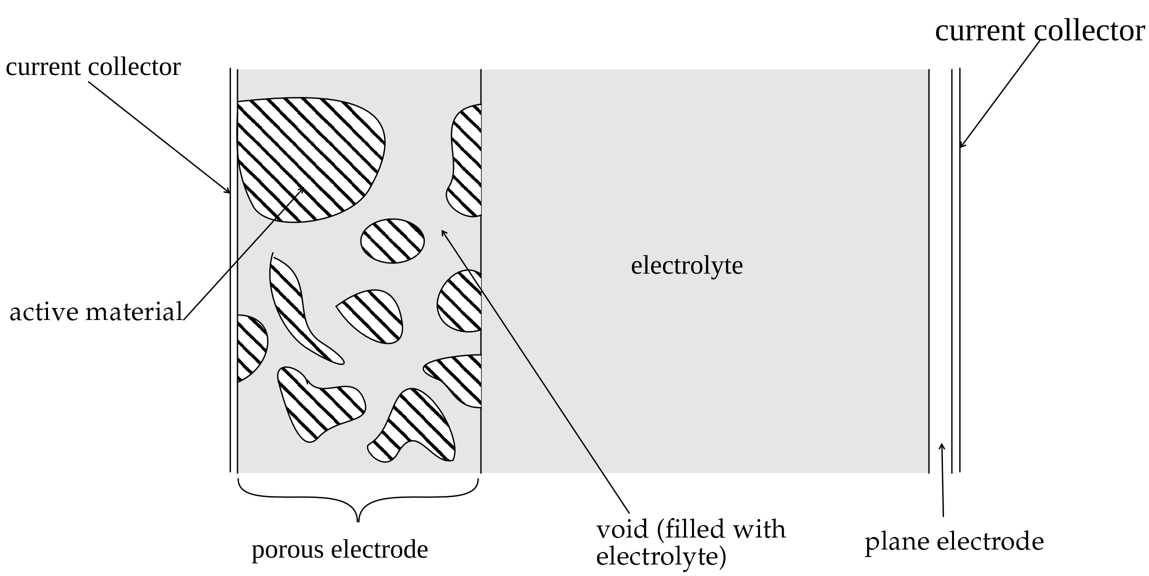


Figure C.9 Schematic of cell showing porous and plane electrodes

- A large interfacial surface area per volume which will increase the rate of the electrochemical reaction.
- Closer proximity of reactants to electrode surface.
- Smaller resistive drop due to shorter current paths.

Porous electrodes may be either matrices of an electrically conductive reactant or of a non-conductive reactant mixed with a conductor. The voids of the matrix are occupied by the electrolytic solution of the cell. The relative diffusion coefficients, size of particles and interstices, and the reaction process itself determines how uniformly the reaction takes place throughout the electrode.

For the purposes of this dissertation, the most significant difference between planar and porous electrodes is the increased surface area available for the reaction. For a planar electrode, the reaction can take place only at the interface between the electrode and the separator. For a porous electrode, depending upon its construction and the relative conductance of the electrolyte and the solid matrix, the reaction may occur throughout the entire electrode simultaneously or in a narrow region which moves through the electrode during discharge [52].

The porous electrode can be characterized by several average quantities. The first is the void volume, the average fraction of the volume that is void. The second is the average surface area of the matrix per unit volume. The electrolyte is assumed to fill the entire void volume of the electrode, so that all of the surface area of the matrix is in contact with the electrolyte. Equations governing electrode processes involving planar electrodes can be applied to porous electrodes so long as effective values of variables based upon the void volume and surface area per volume are used.

Appendix D

Glossary

Activation polarization: Voltage drop due to rate of electrochemical reaction required to provide the load current. Also referred to as surface overpotential.

Active materials: The reactants in a battery's electrochemical reactions.

Amdahl's Law: The performance increase due to a modification is limited by how often that modification may be used.

Anion: A negatively charged ion.

Anode: The electrode in a cell at which oxidation occurs. During discharge, the anode is the negative electrode.

Battery: Two or more electrochemical cells connected in either series or parallel. Commonly used to refer to a cell.

BEST model: Battery Energy Storage Test model for lead-acid cells. Based upon curves of voltage versus current at different depths of discharge.

C rate: The discharge rate normalized to the battery's charge capacity.

Capacity: The amount of charge or energy delivered by a fully charged battery. Typically given in Amp-hours (charge capacity) or Watt-hours (energy capacity).

Cathode: The electrode in a cell at which reduction occurs. During discharge, the cathode is the positive electrode.

Cation: A positively charged ion.

Cell: The basic electrochemical component, consisting of two electrodes and an electrolyte.

Concentration polarization: Voltage drop due to differences in reactant concentrations. Also referred to as concentration overpotential.

Constant current discharge: A discharge where the load's current does not vary.

Constant power discharge: A discharge where the load's power does not vary.

Constant resistance discharge: A discharge where the load's resistance does not vary.

CPU speed-setting: Dynamically modifying the operating frequency of a CPU to reduce energy consumption while still meeting the performance expectations of the user.

Cutoff voltage: Voltage at which a battery is considered fully discharged.

Depth of discharge: The ratio of the amount of charge removed from a battery to its charge capacity.

Discharge rate: The rate at which charge is removed from the battery. Typically given in Amperes, or normalized to the battery's capacity.

Doyle's model: A first principles, finite-mesh model of lithium-ion cells.

Electrode: The location at which an electrochemical reaction occurs.

Electrolyte: The material that enables ions to flow between electrodes in a battery.

Energy: The ability to do useful work, as in moving a charge or a mass. Typically given in Joules or Watt-hours.

Excess cycles: In Weiser et al.'s *Past* policy, the number of cycles of computation left over from a previous window which must be completed in the current window.

Gas-gauge IC: An integrated circuit that monitors the flow of charge into and out of a battery, as well as the battery's temperature, voltage, and other characteristics.

Inflection point: A point on a curve where the slope is 0 but which is not a minima or maxima.

Internal resistance: The opposition to the flow of current within a battery.

KiBaM: Kinetic Battery Model. Models a battery as two wells of charge, an available well and a bound well.

Performance: For this dissertation, the number of iterations of a loop of code that a system can execute per unit time.

Peukert's formula: A battery model for continuous discharges where $capacity = k/current^{\alpha}$.

Power: The rate at which energy is consumed, typically given in Watts.

Power management: Placing idle subsystems into low power modes.

Power-performance trade-off: Scaling back performance to thereby reduce power.

Ragone plot: A figure showing the energy capacity versus power for a battery. Typically a logarithmic plot.

Recovery: A phenomenon whereby a battery may deliver more capacity for an intermittent load than for a continuous load of the same value.

Specific energy: The amount of energy per unit weight of a cell.

Specific power: The amount of power per unit weight of a cell.

Speedup: The ratio of the execution time after a modification to the execution time before a modification.

Standard potential: The voltage of a reactant measured relative to a reference material, typically hydrogen.

State of charge: The amount of charge remaining in a battery.

System power: The total power drawn from a battery by a mobile computer.

Variable-voltage CPU: A CPU that scales its operating voltage with its operating frequency.

Voltage profile: A plot of battery voltage versus depth of discharge.

Work ratio: The number of computations per discharge at a given clock frequency normalized to the computations per discharge at an initial clock frequency.

Y-intercept: The y-coordinate of a line where it crosses the y-axis.

Appendix E

Source code listings

This appendix contains the pertinent portions of the Linux kernel code to change the StrongARM SA-1100 clock, as described in Chapter 4.

```
/*
 *  clock-patch.S Copyright (c) 1998 Tom Martin
 *
 * Assembly routine to change DRAM config registers.
 * Note that this is not meant to be part of the kernel,
 * but should instead be burned into FLASH.
 * Will hardcode the FLASH offset (0x0007ff00)in the kernel.
 *  Should also have the kernel check that this code is present before
 *  jumping to it.
 *
 *  Assemble with arm-unknown-linuxaout-as.  Note that as puts 20 bytes
 * before the actual code, so the kernel should jump to 0x0007ff20
 * if this is burned into 0x0007ff00.
 *
 * Inputs:  r3: MDCNFG, r4: MDCAS0, r5: MDCAS1, r6: MDCAS2.
      r2: &MDCNFG.
 *    lr has return location.
 */
   .text
   .align

   strr3, [r2]
   str r4, [r2, #0x04]
   strr5, [r2, #0x08]
   strr6, [r2, #0x0c]
   movpc, lr
```

```
/*
 *  clock-dram-setup.S Copyright (c) 1998 Tom Martin
 *
 * Assembly routine to change DRAM config registers.
 *  Jumps to routine clock-patch burned into FLASH just before
 *  params location:  0x0007ff00.
 *
 *
 * Inputs:  r0: ptr to array containing new mdcnfg, mdcas[0-2] values.
 *    r1 jump_addr, r2 has MDCNFG’s virtual address.
 *    lr has return location.
 */


   .text
   .align
   .globl _clock_dram_setup
_clock_dram_setup:
   stmfdsp !,{r3 - r6, lr}
   ldr r4, [r0, #0x04]
   ldrr5, [r0, #0x08]
   ldrr6, [r0, #0x0c]
   ldrr3, [r0]
   ldrlr, =_jump_back
   mov pc, r1
_jump_back:
   ldmfdsp!, {r3 - r6, pc}
```

```
/*
 *
 * linux/arch/arm/drivers/char/clock.c
 * Code to control the SA-1100 clock.
 * Copyright (c) 1998, 1999 Tom Martin
 *
 */


/* From <asm/arch/hardware.h> */
/* comments pertain to itsy. otherwise just static memory bank mappings
*/
#define FLASH0                  0x00000000 /* mirror of flash 1 or flash
2 */
#define FLASH1                  0x08000000 /* mother board flash */
#define FLASH2                  0x10000000 /* daughter card flash */
#define FLASH3                  0x18000000 /* unused */



int coproc_switching_disabled = 0;

int speed_values[11]={59, 74, 88, 103, 118, 133, 148, 162, 177, 192,
206};



/*  static_memory_setup needs some fixes:
   --Will get the bank 0 wrong in some cases if there is a
   daughtercard but bank 0 mirrors bank 1.  Should work fine as
   long as bank 0 mirrors bank 2 when there’s a daughtercard.
   --Assumes FLASH is 4 MB.
*/

static int static_memory_setup(int ccf)
{
unsigned int msc0_new, msc0_old, msc1_new, msc1_old;
int config;
int mirror_flag=0;

/*  This shouldn’t be hardcoded.  */
int nonvol_memory_id_address = 0x003fffe0;

/*  Should make the following def’s in clock.h */
int msc_bank_mask = 0x0000ffff;

/*
 *  Assumptions:  Static memory 1 is never invalid because it’s on
 *  the motherboard.  Static memories are 4 MB.  The latter is a
 *  Bad Assumption, but works while I get the kinks out.  (Actually,
 *  the FLASH driver assumes they’re 4 MB too...)
 */

   msc0_old = MSC0;
   msc1_old = MSC1;

   /* Static memory bank 0 mirrors either static memory bank 1 or
```

```
      static memory bank 2.  See Itsy User's Manual.  */
   if ((msc0_old & msc_bank_mask) == (msc1_old & msc_bank_mask)) {
      mirror_flag= 2;
   }
   /* sanity check */
   else if ((msc0_old & msc_bank_mask)!=((msc0_old>>16) &
msc_bank_mask)) {
         printk(" *** static_memory_setup: mirror is wrong\n");
   }
   else {
      mirror_flag = 1;
   }

   config = ((volatile Word *) VirtAdd
(FLASH1+nonvol_memory_id_address))[ccf/2];
   if (ccf % 2) {
      config = config & msc_bank_mask;
   }
   else {
      config = config >> 16;
   }


   msc0_new = config << 16;
   if (mirror_flag == 1) {
      msc0_new |= config;
   }

   if ((msc1_old & msc_bank_mask ) != MSC_INVALID) {
      config = ((volatile Word *) VirtAdd
(FLASH2+nonvol_memory_id_address))[ccf/2];
      if (ccf % 2) {
               config = config & msc_bank_mask;
        }
        else {
               config = config >> 16;
      }
      msc1_new = config;
      if (mirror_flag == 2) {
         msc0_new |=config;
         }
   }
   else msc1_new = MSC_INVALID;

   if ( (msc1_old >>16 )  != MSC_INVALID) {
      config = ((volatile Word *) VirtAdd
(FLASH3+nonvol_memory_id_address))[ccf/2];
               if (ccf % 2) {
         config = config & msc_bank_mask;
      }
      else {
         config = config >> 16;
      }
      msc1_new |= config << 16;
   }
   else  {
      msc1_new |= (MSC_INVALID << 16);
```

```
   }

#ifdef CLOCKDEBUG
   printk(“msc0_new: %x msc1_new: %x \n”, msc0_new, msc1_new);
#endif /* CLOCKDEBUG */

   MSC0 = msc0_new;
   MSC1 = msc1_new;
   return 0;
}
/*  returns 0 if valid speed, -1 if not. */
/*  clobbers r0, r1, r2, r3, r4.  Not sure if the compiler takes care of
that
or not.  */
static int clock_change(int speed)
{
extern void clock_dram_setup(int *, unsigned long , volatile Word *);
int retval, ccf, ppcr_val, ppcr_upper, new_is_faster;
int mdcnfg_new, mdcnfg_old, mdcas0, mdcas1, mdcas2, de_field;

int md_array[4];
ulong jump_addr, flags;


typedef struct {
  int mdcnfg;
  int mdcas0;
  int mdcas1;
  int mdcas2;
} dram_table_t;

dram_table_t *dram_config_addr;

/*  Make these defines in clock.h */
int ccf_mask = 0x0000001f;

   switch (speed) {
      case 59:
         ccf = 0;
         retval = 0;
         break;
      case 74:
         ccf = 1;
         retval = 0;
         break;
      case 88:
         ccf = 2;
         retval = 0;
         break;
      case 103:
         ccf = 3;
         retval = 0;
         break;
      case 118:
         ccf = 4;
         retval = 0;
```

```
            break;
         case 133:
            ccf = 5;
            retval = 0;
            break;
         case 148:
            ccf = 6;
            retval = 0;
            break;
         case 162:
            ccf = 7;
            retval = 0;
            break;
         case 177:
            ccf = 8;
            retval = 0;
            break;
         case 192:
            ccf = 9;
            retval = 0;
            break;
         case 206:
            ccf = 10;
            retval = 0;
            break;
         default:
            retval = -1;

#ifdef CLOCKDEBUG
   printk(“ *** clock_change in default: speed: %d \n”, speed);
#endif /* CLOCKDEBUG */

            return retval;
            break; /* shouldn’t reach here. */
   }

   ppcr_val = inl(PPCR_V);
   ppcr_upper = ppcr_val & (~ccf_mask);

   ppcr_val &=ccf_mask;

   if (ccf > ppcr_val)  {
      new_is_faster = 1;
   }
   else if (ccf < ppcr_val) {
      new_is_faster = 0;
   }
   else { /* The new speed is the same as the old */
      retval = 0 ;
      return retval;
   }

   ppcr_val = ppcr_upper | ccf;

#ifdef CLOCKDEBUG
```

```
        printk(" *** clock_change: speed: %d ppcr = %x\n", speed,
ppcr_val);
#endif /* CLOCKDEBUG */

/*  try using powermgr for now. Doesn't look like it takes
care of the DMA controller however.  */

#ifdef USE_POWERMGR

   {
   int delay;
      if (powermgr_suspend_check(0, &delay)) {
         powermgr_suspend();
   }
   else {
      printk("suspend_check failed with delay %d\n", delay);
      return -EIO;
      }
   }

#endif   /* USE_POWERMGR */


   /*  Disable D-cache and WB */
/* asm("mrcp15, 0, r1, c1, c0");
   asm("bicr1, r1, #0x000c");
   asm("mcrp15, 0, r1, c1, c0");
*/

   mdcnfg_old  = (MDCNFG);
   de_field = mdcnfg_old & DE_FLD_MASK;

/*
   if (de_field > 1 ) {
      mdcnfg_new = mdcnfg_values[ccf+11][0] | de_field;
      mdcas0 = mdcnfg_values[ccf+11][1];
      mdcas1 = mdcnfg_values[ccf+11][2];
      mdcas2 = mdcnfg_values[ccf+11][3];
   }
   else {
      mdcnfg_new = mdcnfg_values[ccf][0] | de_field;
                mdcas0 = mdcnfg_values[ccf][1];
                mdcas1 = mdcnfg_values[ccf][2];
                mdcas2 = mdcnfg_values[ccf][3];
        }
*/

   /* From Debby's email, 11/18/1998 */
   dram_config_addr = (dram_table_t *)VirtAdd(0x000010cc);

   mdcnfg_new = (dram_config_addr+ccf)->mdcnfg | de_field;
        mdcas0 = (dram_config_addr+ccf)->mdcas0;
        mdcas1 = (dram_config_addr+ccf)->mdcas1;
        mdcas2 = (dram_config_addr+ccf)->mdcas2;

   md_array[0] = mdcnfg_new;
```

```
   md_array[1] = mdcas0;
   md_array[2] = mdcas1;
   md_array[3] = mdcas2;
   jump_addr = VirtAdd(FLASH1 + CLOCK_PATCH +0x00000020);

   /*  Test to make sure the patch is in FLASH.  */
   if ( *(unsigned int*)(jump_addr) != 0xe5823000 )  {

      printk("clock_change: DRAM setup patch is not in FLASH!\n");
      return -1;
   }

#ifdef CLOCKDEBUG
   else
      printk("clock_change: DRAM setup patch is in FLASH.\n");


   printk("clock_change, md_array: %x jump_addr: %x &MDCNFG %x\n",
md_array, jump_addr, &MDCNFG);

#endif /* CLOCKDEBUG */

   if ( new_is_faster ) {
      save_flags_cli (flags);

      /*  Putting the call to the pcmcia_setup stub here for now.
      When PCMCIA is working, the stub will have to change
      MECR.  May have to test whether the cards care when
      their timing is changed relative to the CPU.  May also
      have MECR change in powermgr. */

      pcmcia_setup(ccf);

      static_memory_setup(ccf);

#ifdef FLASH_PATCH
      clock_dram_setup(md_array, jump_addr, &MDCNFG);

#else
      MDCAS2 = mdcas2;
      MDCAS1 = mdcas1;
      MDCAS0 = mdcas0;
      MDCNFG = mdcnfg_new;
#endif /* FLASH_PATCH */
      restore_flags(flags);
   }

#ifdef LCD_STATIC
   lcdStatic();
#else
#ifdef LCD_OFF
   lcdOff();
#endif
#endif

#ifndef USE_POWERMGR
```

```
   TurnUartOff(serial_save);
#endif



/*  Might be getting lucky that writing r1 doesn't clobber anything. */
 /*disable clock switching*/
   asm("mcr        p15, 0, r0, c15, c2, 2");
   asm("mov        r1, #0xe8000000");
   asm("add        r1, r1, #0x00050000");
   asm("ldr        r0, [r1]");  /* force a cache miss */
                 /* r1 points to a uncacheable address */

   outl(ppcr_val, PPCR_V);

/* re-enable clock switching  if it hasn't been forced off.*/
/*  Test to see if this needs to be here.  I suspect that changing ppcr
    enables clock switching.  */
   if (coproc_switching_disabled == 0) {
      __asm__ __volatile__("mcreq        p15, 0, r0, c15, c1, 2");
   /* printk("Switching not disabled\n"); */
   }


#ifndef USE_POWERMGR
   TurnUartOn(serial_save);
#endif

   if ( !new_is_faster ) {
      save_flags_cli (flags);

      /* See comment after "if (new_is_faster)" above.*/
      pcmcia_setup(ccf);

      static_memory_setup(ccf);

#ifdef FLASH_PATCH
      clock_dram_setup(md_array, jump_addr, &MDCNFG);
#else
      MDCNFG = mdcnfg_new;
      MDCAS0 = mdcas0;
      MDCAS1 = mdcas1;
      MDCAS2 = mdcas2;

#endif  /* FLASH_PATCH */
      restore_flags(flags);
   }


   /*  Enable D-cache and WB */
/* asm("mrcp15, 0, r1, c1, c0");
   asm("orrr1, r1, #0x000c");
   asm("mcrp15, 0, r1, c1, c0");
*/
```

```
#if  (defined(LCD_STATIC)  || defined (LCD_OFF) )
   lcdOn();
#elif (!defined(USE_POWERMGR))
   lcd_setup(speed);
#endif /* LCD_STATIC || LCD_OFF */

#ifdef  USE_POWERMGR
/* see arch/arm/kernel/sa1100-sleep-mode.c  & powermgr.c */

   powermgr_resume(inl(PSSR_V));

#endif /* USE_POWERMGR */

#ifdef CLOCKDEBUG
   printk(" After powermgr_resume: LCRR3: %x \n", LCCR3);
#endif /* CLOCKDEBUG */

   return retval;
}


static inline void sa1100_disable_switching(void)
{
   __asm__ __volatile__("mcr        p15, 0, r0, c15, c2, 2");  /*disable
clock switching*/
   __asm__ __volatile__("mov        r1, #0xe8000000");
   __asm__ __volatile__("add        r1, r1, #0x00050000");
   __asm__ __volatile__("ldr        r0, [r1]");  /* force a cache miss */
      /* r1 points to a uncacheable address */

}


static inline void sa1100_enable_switching(void)
{
   asm("mcr        p15, 0, r0, c15, c1, 2"); /*renable clock switching*/
}

static inline int clock_full(void)
{
   sa1100_enable_switching();
   coproc_switching_disabled = 0;
   return 0;
}
```

Bibliography

[1] Agarwal, A. *Analysis of cache performance for operating systems and multiprogramming*. Boston: Kluwer Academic Publishers, 1989.

[2] Anderson, J., Berc, F., Dean, J., Ghemawat, S., Henzinger, M., Leung, S., Sites, R., Vandevoorde, M., Waldspurger, C., Weihl, W. “Continuous profiling: Where have all the cycles gone?” ACM Transactions on Computer Systems, vol.15, no.4, November 1997, p. 357-90.

[3] Athas, W., Svennson, L. Koller, J., Tzartzanis, N. and Chou, E. “Low-Power Digital Systems Based on Adiabatic-Switching Principles,” IEEE Transactions on VLSI Systems, December 1994, pp. 398-407.

[4] Benchmarq Microelectronics, Inc. 1994 Data Book.

[5] Bennett, H. “Logical Reversibility of Computation,” IBM Journal of Research and Development, November 1973, pp. 525-532.

[6] Bennett, H. and Landauer, R. “The Fundamental Physical Limits of Computation,” Scientific American, July 1985, pp. 48-56.

[7] Burd, T. and Brodersen, R. “Energy Efficient CMOS Microprocessor Design,” Proceedings of the Twenty-Eighth Annual Hawaii International Conference on System Sciences; Vol.1; Wailea, HI, January 1995.

[8] Bhattacharya, D. “Power Management White Paper,” Intel Corporation, Rev. 0.2.

[9] Chandrakasan, A. and Brodersen, R. *Low Power Digital CMOS Design*. Boston: Kluwer Academic Publishers, 1995.

[10] Chang, J. and Pedram, M. “Battery-powered digital CMOS design,’’ Procings of Design Automation and Test in Europe, March 1999, pp. 72-76.

[11] Child, J. “Making Every Watt Count,” Computer Design, December 1993, vol. 32, no. 12, pp. 67-70, 83-86.

[12] Chemical Rubber Company. *CRC Handbook of Chemistry and Physics*. 76th edition. CRC Press, Cleveland, OH, 1995.

[13] De Vidts, P. and White, R.E. “Mathematical modeling of a nickel-cadmium cell: proton diffusion in the nickel electrode,” Journal of the Electrochemical Society; vol.142, no.5; May 1995; pp. 1509-19.

[14] Digital Equipment Corporation. DIGITAL Semiconductor SA-1100 Microprocessor: Technical Reference Manual, revision EC-R5MTC-TE, March 1998.

[15] Doyle, M. Design and Simulation of Lithium Rechargeable Batteries. Ph.D. Dissertation, University of California at Berkeley. 1995.

[16] Doyle, M., Fuller, T. and Newman, J. "Modeling of Galvanostatic Charge and Discharge of a Lithium/Polymer/Insertion Cell," Journal of the Electrochemical Society; vol. 140, no. 6; June 1993; pp. 1526-1533.

[17] Doyle, M. and Newman, J. "Analysis of Capacity-Rate Data for Lithium Batteries Using Simplified Models of the Discharge Process," to appear in the Journal of Applied Electrochemistry.

[18] Doyle, M. and Newman, J. "Comparison of Modeling Predictions with Experimental Data from Plastic Lithium Ion Cells," Journal of the Electrochemical Society; vol. 143, no. 6; June 1996; pp. 1890-1903.

[19] Doyle, M., Newman, J. and Reimers, J. "A quick method for measuring the capacity versus the discharge rate for a dual lithium-ion insertion cell undergoing cycling," Journal of Power Sources; vol. 52, no. 2; December 1994; pp. 211-216.

[20] Elnozahy, E. Private communication.

[21] Fan, D. and White, R.E. "Mathematical Modeling of a Nickel-cadmium Battery. Effects of Intercalation and Oxygen Reactions," Journal of the Electrochemical Society; vol.138, no.10; Oct. 1991; pp. 2952-60.

[22] Feynman, R. "Quantum Mechanical Computers," Foundations of Physics, Vol. 16, No. 6, 1986, pp. 507-531.

[23] Flinn, J. and Satyanarayanan, M. "PowerScope: A Tool for Profiling the Energy Usage of Mobile Applications," 2nd IEEE Workshop on Mobile Computing Systems and Applications, New Orleans, LA, February 25-26, 1999, pp. 2-10.

[24] Flynn, J. "How to Talk to 'Smart Batteries'," Embedded Systems Programming, vol. 9, no. 12, November 1996, pp. 30-46.

[25] Fuller, T., Doyle, M. and Newman, J. "Relaxation Phenomena in Lithium-Ion-Insertion Cells,", Journal of the Electrochemical Society; vol. 141, no. 4; April 1994; pp. 982-990.

[26] Fuller, T., Doyle, M. and Newman, J. "Simulation and Optimization of the Dual Lithium Ion Insertion Cell," Journal of the Electrochemical Society; vol. 141, no. 1; January 1994; pp. 1-10.

[27] Ganssle, J. “A Plea to Compiler Vendors,” Embedded Systems Programming, vol. 12, no. 3, March 1999, pp. 129-132.

[28] Gates Energy Products, Inc. Rechargeable Batteries Applications Handbook. Boston: Butterworth-Heinemann, 1992.

[29] Glass, B. “Under the Hood: Power Management,” Byte, September 1991, pp. 329-335.

[30] Goodenough, J. “Design Considerations,” Solid State Ionics 69, 1994, pp. 184-198.

[31] Govil, K., Chan, E., and Wasserman, H. “Comparing Algorithms for Dynamic Speed-Setting of a Low-Power CPU,” Proceedings of the 1st ACM International Conference on Mobile Computing and Networking, 1995, pp. 13-25.

[32] Graham, S., Kessler, P., McKusick, M. “gprof: A Call Graph Execution Profiler,” Proceedings of the SIGPLAN ‘82 Symposium on Compiler Construction, SIGPLAN Notices, June 1982, Vol. 17, No. 6, pp. 120-126.

[33] Hageman, S. “Simple PSpice Models Let You Simulate Common Battery Types,” EDN, October 28, 1993, pp. 117-129.

[34] Horowitz, M, Indermaur, T, and Gonzalalez, R. “Low-Power Digital Design,” Proceedings of the 1994 Symposium on Low Power Electronics, October 1994, pp. 8-11.

[35] Hyman, E., Spindeler, W.C., and Fatula J. F. “Phenomenological Discharge Voltage Model for Lead-Acid Batteries,” Proceedings of the AIChE Meeting, November 1986, pp. 78-86.

[36] Intel Corporation, Microsoft Corporation. “Advanced Power Management: The Next Generation,” Version 1.0, August 18, 1991.

[37] Intel Corporation. Memory Products Databook, 1993.

[38] Keyes, R. “Physical Limits in Digital Electronics,” Proceedings of the IEEE, Vol. 63, May 1975, pp. 740-767.

[39] Keyes, R. and Landauer, R. “Minimal Energy Dissipation in Logic,” IBM Journal of Research and Development, March 1970, pp. 152-157.

[40] Kiaei, S. and Devadas, S. Panel session organizers, “Which has greater potential power impact: High-level design and algorithms or innovative low power technology?” Proceedings of the 1996 International Symposium on Low Power Electronics and Design, August 1996, pp. 175.

[41] Koller, J. and Athas, W. "Adiabatic Switching, Low Energy Computing, and the Physics of Storing and Erasing Information," Proceedings of the Workshop on Physics and Computation, October 1992,

[42] Kuroda, T. Suzuki, K., Mita, S., Fujita, T., Yamane, F., Sano,F., Chiba,A., Watanabe, Y., Matsuda, K., Maeda, T., Sakurai, T., and Furuyama, T. "Variable supply-voltage scheme for low-power high-speed CMOS digital design," IEEE Journal of Solid-State Circuits, Vol. 33, No. 3, March 1998, pp. 454-462.

[43] Linden, D. *Handbook of Batteries and Fuel Cells*. New York: McGraw-Hill, 1984.

[44] Lindstrom, O. "A Pseudo-Resistance Method for Collating Battery Discharge Data," Journal of the Electrochemical Society; Vol. 117, No. 8; August 1970; pp. 1083-1090.

[45] Lorch, J. and Smith, A. "Software strategies for portable computer energy management," IEEE Personal Communications Magazine, Vol. 5, No. 3, June 1998, pp. 60-73.

[46] Manwell, J. and McGowan, J. "Lead Acid Battery Storage Model for Hybrid Energy Systems," Solar Energy Vol. 50, No. 5, pp. 399-405.

[47] Martin, T. "Evaluation and Reduction of Power Consumption in the Navigator Wearable Computer", Masters Thesis, Carnegie Mellon University Department of Electrical and Computer Engineering, 1994.

[48] Martin, T. and Siewiorek, D. "A Power Metric for Mobile Systems," Proceedings of the 1996 International Symposium on Low Power Electronics and Design, August 1996, pp. 37-42.

[49] Mayer, J. "Design Team Gives Thumbs Up to Portable Platform," Portable Design, February 1996, pp. 47-51.

[50] Mead, C. and Conway, L. *Introduction to VLSI Systems*. Reading, MA. Addison-Wesley Publishing Company, 1980.

[51] NEC Electronics Inc. VR4100 Microprocessor Product Brief, 1995.

[52] Newman, J. *Electrochemical Systems*, 2nd Edition. Englewood Cliffs, NJ: Prentice Hall, 1991.

[53] Newman, J. "Optimization of Porosity and Thickness of a Battery Electrode by Means of a Reaction-Zone Model," Journal of the Electrochemical Society; vol 142, no. 1; January 1995; pp. 97-101.

[54] Ong, P. and Yan, R. “Power-Conscious Software Design--a framework for modeling software on hardware,” Proceedings of the 1994 Symposium on Low Power Electronics, October 1994, pp. 36-37.

[55] Paleologo, G., Benini, L., Bogliolo, A., and De Micheli, G. “Policy Optimization for Dynamic Power Management,” Proceedings of the 35th Design Automation Conference, San Francisco, CA, June 15-19, 1998, pp. 182-187.

[56] PC Magazine, “Travelling powerhouses,” January 19, 1999.

[57] PC Magazine, “Notebook computers,” July 20, 1999.

[58] Pedram, M., Tsui, C-Y. and Wu, Q. “An integrated battery-hardware model for portable electronics,’’ , Proc. of Asia and South Pacific Design Automation Conf., February 1999, pages 109-112.

[59] Pering, T. and Brodersen, R. “Energy Efficient Voltage Scheduling for Real-Time Operating Systems,” presented at the Fourth IEEE Real-Time Technology and Applications Symposium, Works in Progress session, Denver, CO, June 3-5, 1998.

[60] Petersen, R. *Linux: The Complete Reference.* Berkeley, CA: Osborne McGraw-Hill, 1996.

[61] Podlaha, E. and Cheh, H. “Modeling of Cylindrical Alkaline Cells,” Journal of the Electrochemical Society, vol. 141, no. 1; January 1994; pp. 28-35.

[62] Sanyo Corporation. UF812248 Data Sheet. April, 1998.

[63] Sanyo Corporation. UF612248 Data Sheet. April, 1998.

[64] Seymour, J. “386SX laptops: Desktop power, notebook size,” PC Magazine, vol. 10, no. 14, August 1991, pp. 103-230.

[65] Smailagic, A. and Siewiorek, D. “The CMU Mobile Computers: A New Generation of Computer Systems,” Proceedings of COMPCON ‘94; San Francisco, CA, 28 February-4 March 1994.

[66] Sony Corporation, Recording Media and Energy Company. Lithium Ion Rechargeable Battery Data Sheet, 1996.

[67] Su, C., Tsui, C., and Despain, A. “Low Power Architecture Design and Compilation Techniques for High-Performance Processors,” Proceedings of COMPCON ’94, pp. 489-498.

[68] Tiwari, V., Malik, S., and Wolfe, A. "Compilation Techniques for Low Energy: An Overview," Proceedings of the 1994 Symposium on Low Power Electronics, October 1994, pp. 38-39.

[69] Tiwari, V., Malik, S., and Wolfe, A. "Power Analysis of Embedded Software: A First Step Towards Software Power Minimization," IEEE Transactions on Very Large Scale Integration Systems, vol. 2, no. 4, December 1994, pp. 437-445.

[70] Uhlig, R., Nagle, D., Mudge, T., Sechrest, S., and Emer, J. "Instruction Fetching: Coping with Code Bloat," Proceedings of the 22nd International Symposium on Computer Architecture, July 1995, pp. 345-356.

[71] Viredaz, M. "The Itsy Pocket Computer Version 1.5 User's Manual," Compaq Western Research Laboratory Technical Note TN-54, July 1998.

[72] Weiser, W., Welch, B., Demers, A., and Shenker, S. "Scheduling for Reduced CPU Energy," Proceedings of the 1st USENIX Symposium on Operating Systems Design and Implementation, November 1994, pp. 13-23.

[73] Wuytack, S., Francky, C., Franssen, F., Nachtergaele, L., and De Man, H. "Global communication and memory optimizing transformations for low power systems," Proceedings of the 1994 Workshop on Low Power Design, April 1994, pp. 203-208.

[74] Yao, F., Demers, A. and Shenker, A. "A Scheduling Model for Reduced CPU Energy," Proceedings of the 36th Annual Symposium on Foundations of Computer Science, October 1995, pp. 374-382.

[75] Younis, S. and Knight, T. "Asymptotically Zero Energy Split-Level Charge Recovery Logic," Proceedings of the 1994 International Workshop on Low Power Design, April 1994, pp. 177-182.

[76] Zorzi, M. and Rao, R. "Error control and energy consumption in communications for nomadic computing," IEEE Transactions on Computers, Vol. 46, No. 3, March 1997, pp. 279-289.