The Impact of Battery Capacity and Memory Bandwidth on CPU Speed-Setting: A Case Study

Thomas L. Martin, Daniel P. Siewiorek
Institute for Complex Engineered Systems
Carnegie Mellon University
Pittsburgh, PA 15213
tlm@cs.cmu.edu, dps@cs.cmu.edu

1. ABSTRACT

The purpose of this paper is to report the power and performance of an application on a real system as the CPU frequency varies. Previous work in CPU speed-setting considered only the power of the CPU and only CPU's that vary supply voltage with frequency. This work takes a broader approach, considering total system power, battery capacity and main memory bandwidth. The results, which are up to a factor of four less than ideal, show that all three must be considered when setting the CPU speed, whether the speed is fixed at a single value or varied dynamically during operation.

2. INTRODUCTION

Technical developments in recent years have enabled designers to build hand-held and wearable computers. In many of these computers, the CPU consumes a substantial fraction of the total power, making the CPU a prime target for power savings. The purpose of this paper is to report the battery discharge measurements of an application on a real system as the CPU frequency is varied. The goal of the research was to decide whether the Itsy, a StrongARM SA-1100-based hand-held computer from Compaq's Western Research Lab [17], would benefit from having a dynamic CPU speed-setting policy, and if not, to determine the fixed speed at which the CPU should be operated to balance power and performance. The paper examines the power, performance, and battery life of the Itsy while running an MPEG video player over a range of CPU frequencies. The experimental results are up to nearly a factor of four less than what would be expected given ideal assumptions about the system's battery capacity and performance.

The remainder of this paper is organized as follows: Section 3 describes the related work in the area of CPU speed-setting. Section 4 describes the three major factors in CPU speed-setting from a system perspective. Section 5 details the experimental method. Section 6 presents the results of the experiments, showing the impact of the factors described in Section 4. Finally, Section 7 explains how these results affect CPU speed-setting policies in general and outlines future work in the area.

3. RELATED WORK

Several researchers have investigated saving power by dynamically setting the clock frequency of the CPU. Weiser et al. first pointed out that it is now feasible to build systems that can use dynamic speed-setting, varying CPU voltage and frequency on-the-fly, to reduce energy consumption [18]. The speed would be set by the operating system to reduce the energy consumption of the CPU while still meeting the performance requirements of the user. Weiser et al. and Govil et al. [8] used trace-driven simulation to study several dynamic speed-setting policies that slowed the CPU when it was mostly idle and sped it up when it was mostly active. Yao et al. proposed a scheduling model for dynamic speed-setting and compared some on-line algorithms to an off-line algorithm that computed a minimum energy schedule [20]. Pering and Brodersen applied the concept of real-time scheduling to dynamic speed-setting [13]. All of these researchers assumed that the CPU power was proportional to s^3, where s is the factor by which the CPU frequency and voltage are changed. This assumption is based upon the dynamic power of CMOS devices being equal to CV^2f, where C is the effective switching capacitance, V is voltage, and f is the frequency [19]. Because the maximum frequency of the device is limited by its voltage, it is not possible to lower the voltage without also lowering the frequency. In addition to assuming power proportional to s^3, they assumed the execution time of an operation to be inversely proportional to the clock frequency, or $1/s$. Given these assumptions, the energy per operation is proportional to s^2 and is minimized by running at the slowest CPU frequency. Furthermore, based upon these assumptions, they concluded that it would not be beneficial to reduce the CPU frequency without also reducing the voltage, because in this case the energy per operation would be constant.

This paper takes a more general approach. We consider the energy consumption of the entire system rather than the consumption of just the CPU, and take into account the loss of battery capacity with increasing power. We measure the performance of an application on a real system over a range of CPU frequencies and the number of computations that the system completes during a battery discharge. The results show that even with a fixed-voltage CPU, there are situations where running at less than its maximum frequency is advantageous.

4. SYSTEM FACTORS IN SPEED-SETTING

When the performance expectations of a user are met by a mobile computer, the next major consideration should be the number of computations per battery discharge. The computations per discharge is essentially the battery capacity divided by the energy per operation for a given computation. From a system perspective, the three major factors in setting the CPU speed to increase the computations per discharge are 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. The relationship between these factors and the number of computations per discharge is given by

$$\textit{Computations per Discharge} = \frac{\textit{Capacity(SystemPower(f))} \times \textit{Performance(f)}}{\textit{SystemPower(f)}}$$

where *f* is the CPU clock frequency, and the battery capacity is given in Watt-hours, the power in Watts, and the performance in iterations per hour.

The first factor, battery capacity, is typically assumed to be constant. When battery capacity is constant, maximizing computations per discharge is the same as minimizing the energy per operation. In reality, however, battery capacity decreases as the load power increases, typically by as much as 20%-40% over a useful range of load power [10]. In situations where battery capacity is not constant, minimizing the energy per operation may not maximize the computations per discharge. We showed previously the impact of loss of battery capacity on computations per discharge using Peukert's equation, a first-order, empirical battery model for continuously-on loads [11]. Since most mobile computers are used intermittently rather than continuously, we looked for a model capable of calculating battery capacities for intermittent discharges. Doyle and Newman implemented a one-dimensional, finite-difference model for Li-ion cells, which has been used by their industrial partners to study the effect of cell design parameters and to verify a technique for quick determination of the capacity vs. power [5][6]. We used Doyle's model to study the effect of intermittent discharges on the capacity and found that peak power predicts battery capacity better than average power. Figure 1 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

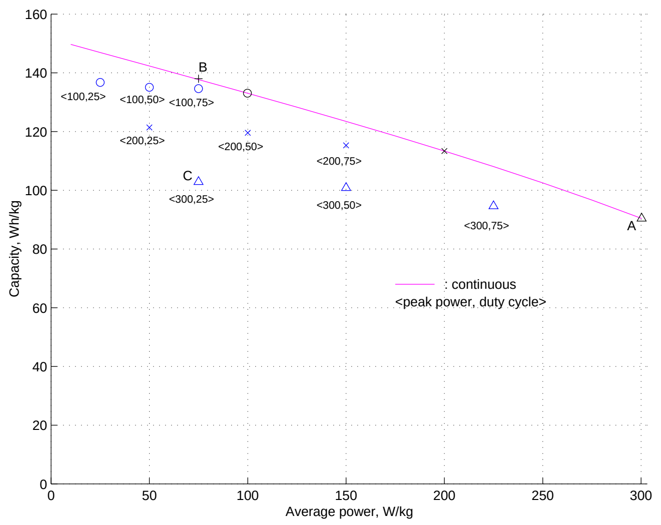

Figure 1. Li-ion model results for capacity versus average power, showing difference between continuous and intermittent loads of same average value

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%, while using the peak power would under-estimate it by only about 10%. To put these results in more common terms, the 75 W/kg continuous load 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 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. Similar results were obtained in experiments with actual Li-ion batteries, so this behavior is not simply a product of the model.

The second factor in setting the CPU speed is the power of the system as a function of frequency. This factor, together with the performance as a function of frequency, determines the CPU frequency which gives the minimum energy per operation. The previous research considered only the CPU power. But the total system power determines the battery life, not just the power of the processor. If a system were to be actually constructed using a variable-voltage CPU, with CPU power proportional to s^3 as described in Section 3, one would expect there would be other terms in the function of system power versus frequency due to subsystems that operate at a fixed voltage, changes in efficiency of the power supply over the range of loads, and other reasons. For example, we curve-fit the power versus frequency data for a variable-voltage CPU [9] and found that even the power of the CPU alone may have lower order terms that amount to about 10% of the total power at the fastest speed. These lower-order terms mean that the slowest CPU frequency may not result in the minimum energy per operation. If the lower-order terms are large enough, running too slowly may increase the energy per operation. Most systems currently available, however, are constructed with a CPU operating at a fixed voltage, with subsystems that operate at frequencies independent of the CPU's frequency. The power of the system is then a linear function of CPU frequency.

The final factor in setting the CPU speed is the performance as a function of frequency. The assumption used in our previous work, in other speed-setting work, and in the previous paragraph's statements about energy per operation is that performance is proportional to frequency [8][11][18][20]. This ignores the memory subsystem. For example, in applications with a high cache miss ratio, performance can be limited by memory bandwidth rather than CPU frequency. Once an application becomes limited by memory bandwidth, increasing the CPU frequency will have little effect on the application performance and will increase the energy per operation.

In the following sections, we study each of these three factors using a system based on a fixed-voltage CPU.

5. METHOD

The experiments described in this paper were performed using the Itsy, a hand-held computer from Compaq's Western Research Laboratory [17]. The Itsy is based on the StrongARM SA-1100 [4] and runs the Linux operating system.

We added two calls to the Linux kernel for changing the CPU frequency based upon the clocking scheme for the StrongARM, which is shown in Figure 2. 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. 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 combination of the two calls allows the CPU core frequency to vary from 29.5 to 206 MHz.

The code chosen for the experiments was the Itsy's MPEG video player. The justification for this choice is twofold. First, we expect a video player to be one of the typical applications for a mobile computer. Second, we wanted

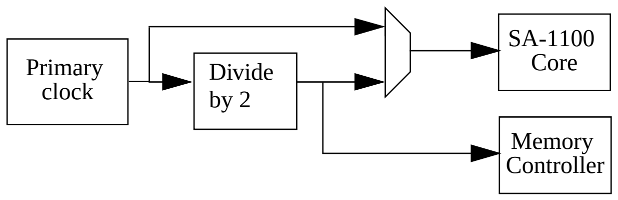

Figure 2. Clocking scheme for the StrongARM SA-1100

code that adequately exercised 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 [16].

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. 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 previous section--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.

6. RESULTS

This section describes the measurements of performance versus frequency, power versus frequency, and iterations per discharge of the Itsy running the MPEG player. The results show that setting the CPU speed depends on all three of the factors discussed in Section 4.

Figure 3 shows the normalized performance versus frequency for the MPEG loop. The measured performance is nearly ideal until about 100 MHz, where there is a breakpoint. From 100 MHz to the maximum frequency of 206 MHz, the measured performance diverges from ideal. Examining the performance of the functions of the MPEG player explains why.

Figure 4 shows the normalized performance versus the CPU frequency for the two functions with the most execution time in the MPEG player. These times were measured using *gprof* [7]. The points show the measured performance, the solid line shows the expected performance given the change

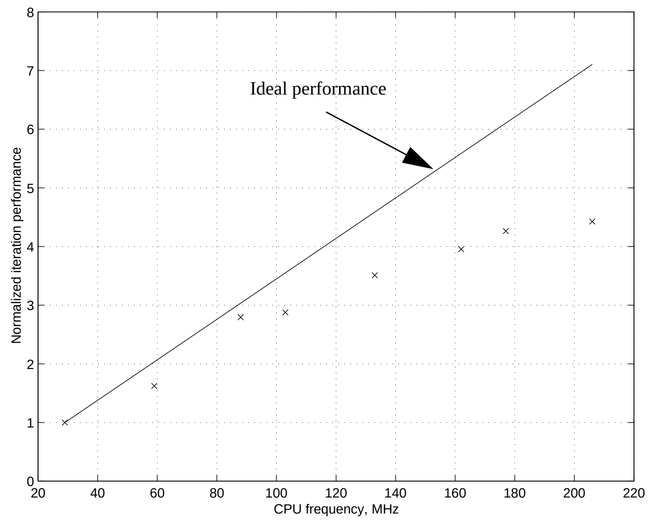

Figure 3. Performance vs. frequency of MPEG loop

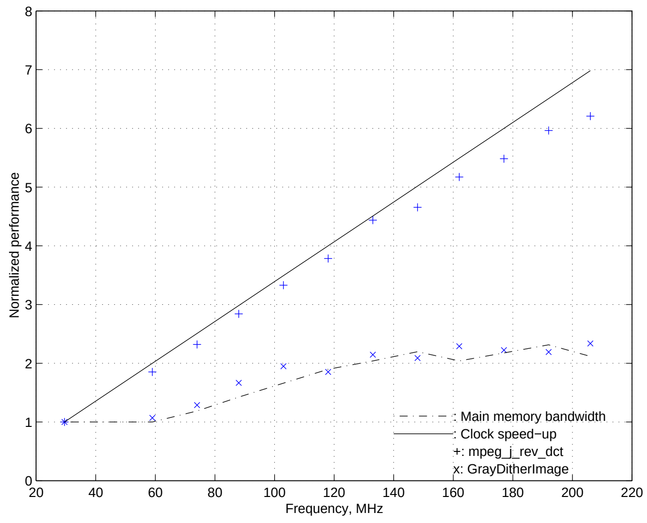

Figure 4. Performance versus frequency for two of the MPEG player functions, with expected speed-up and measured main memory bandwidth

in clock speed, and the dashed line shows the normalized, 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.

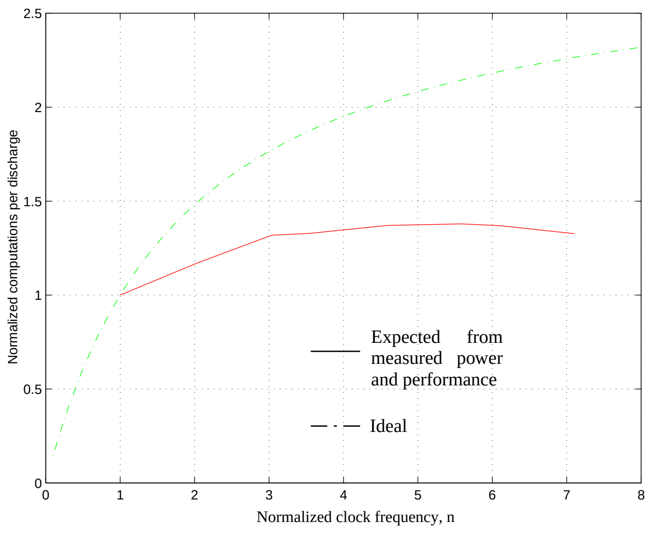

Figure 5. Ideal and expected computations per battery life

Figure 5 shows the expected iterations per discharge calculated from performance and power measurements and the iterations per discharge if the performance speedup 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 [14][15]. The results of the discharge measurements are shown in Figure 6. These cells are nearly ideal over the range of the power of the Itsy. Consequently, the measured iterations per discharge agree with the expected curve quite well, although the measured points begin to be less than expected at the higher frequencies due to loss of battery capacity. Figure 6 shows that ideal assumptions about performance versus frequency can be quite misleading.

But Figure 7 shows that ideal assumptions can be even more misleading when the power of the system lies in the non-ideal range of a battery's capacity. These results were collected 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 [12]. Thus we would expect to see similar results if the experiments were conducted using the notebook computers.

Figure 7 shows that assuming ideal behavior for performance versus frequency and battery capacity versus

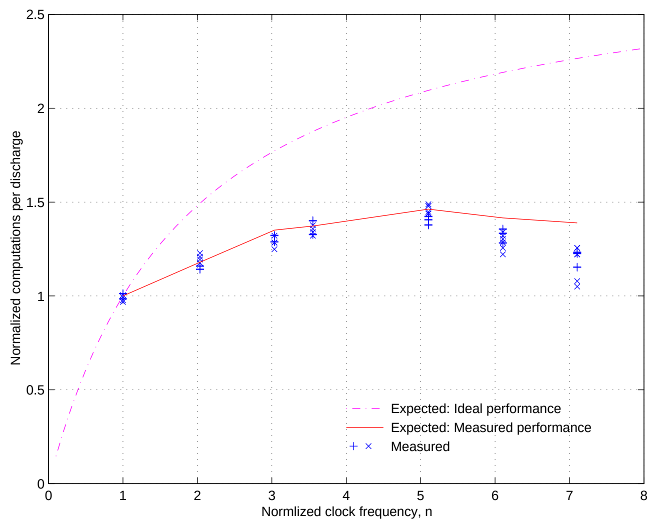

Figure 6. Expected and measured results with Li-ion batteries, Sanyo UF-510 and UF-310

power can be mistaken. The measured iterations per discharge were up to nearly a factor of four less than the iterations per discharge predicted by ideal behavior for both factors. Even using measured performance versus frequency, the iterations per discharge decreased as the clock frequency was increased, rather than increasing as predicted by assuming an ideal battery capacity.

7. CONCLUSIONS

A CPU speed-setting policy for a general purpose system must consider system power, battery capacity and memory bandwidth. This paper has shown that there are regions of operation where ideal assumptions about any of these factors will lead to a decrease in the number of computations performed per discharge cycle. The results shed light on the necessary ingredients for a successful speed-setting policy. For the Itsy, with its fixed-voltage CPU, the speed should be set to the lowest speed that yields

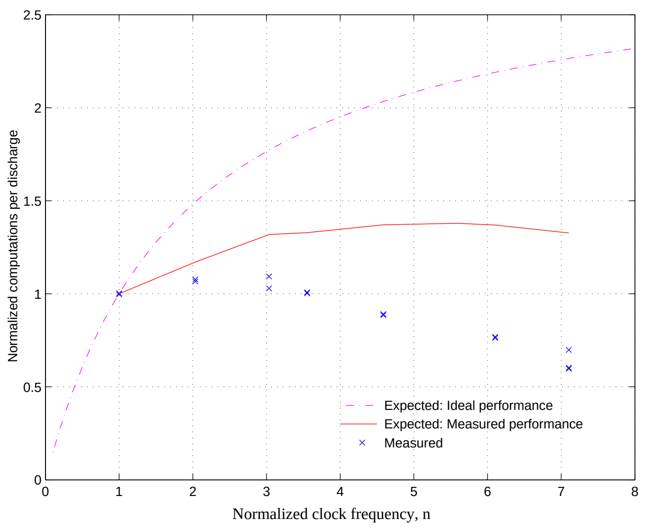

Figure 7. Expected and measured results with AAA batteries

performance acceptable to the user if AAA batteries are the power source. If instead a battery with ideal behavior is the power source, such as the Sanyo Li-ions, the system could benefit from a speed-setting policy that varied from approximately 133 MHz to 206 MHz based upon the performance versus frequency characteristic of the active application.

While we have focused on only the MPEG player, the characteristic of the program upon which the results depend--performance limited by memory bandwidth--is likely to be shared by other applications expected to be typical in a mobile environment. Cache interference of operating system and application code could lead to more main memory bandwidth limitations than would be found in the applications alone. As part of our future work, we intend to study other typical applications. However, the results show that there exist real applications with non-ideal performance behavior that impacts the choice of CPU speed.

The results also shed light on the characteristics of a system built with a variable-voltage CPU. First, the memory subsystem should be a major design consideration, so that it does not limit performance over the range of CPU frequencies. Second, the portion of the system that can vary voltage is important. For example, if the SA-1100 were a variable-voltage CPU, only its 1.5V supply would be able to vary. The 3.3V supply powers the CPU's I/O pins and must remain fixed. Figure 8 shows the measured system power, the measured power on the 1.5V and 3.3V supplies, and an estimate of the total system power if the 1.5V supply were allowed to vary with the frequency. The system power versus frequency would then be of the form $a_3s^3 + a_1s + a_0$. Figure 8 shows that if only the 1.5V CPU voltage is allowed to vary, then the power of the system will be dominated by the 3.3V supply, which supplies the CPU I/O pins and other subsystems. The system power appears to be more linear than cubic. Consequently, even if system performance and battery capacity were ideal, the assumption that the system power is proportional to s^3 would be invalid, and the goal of "running as slowly as possible" this assumption leads to would be invalid as well.

When system performance and battery capacity are non-ideal, they must be accounted for by a speed-setting policy. The desirable features of a system that allows the OS to set the speed may include hints from applications, cache performance registers in the CPU like those found in high-end systems [2], and battery information using battery "gas gauge" integrated circuits [3]. The impact of I/O performance bottlenecks on speed-setting also needs to be considered. Whereas ideally a policy would have the CPU's minimum operating speed as its lower bound, these results have shown that the lower bound should be the speed that maximizes the computations per discharge.

This paper has shown that there are multiple feedback variables for a speed-setting policy. If the OS is to set the CPU speed, it must have adequate feedback about each of

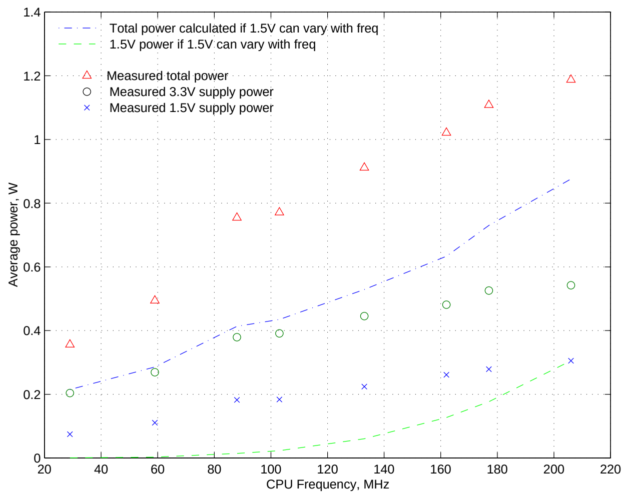

Figure 8. Measured system power, and estimated system power if 1.5V CPU supply could scale with frequency

the three factors discussed in this paper: performance as a function of CPU frequency, system power as a function of CPU frequency, and battery capacity as a function of system power. Ignoring any one of these factors may lead to a decrease in the number of computations performed in a discharge. A realistic policy will take all three factors into account to set its minimum speed to the value that maximizes the computations per discharge.

8. ACKNOWLEDGMENTS

The results with the Itsy would not have been possible without the generous donation of Itsies by Compaq's Western Research Laboratory. We are also indebted to Marc Doyle for giving us access to his Li-ion cell model. Thanks to LeMonté Green for providing helpful comments on the text. The Wearable Computing Project at Carnegie Mellon University is funded by NSF and DARPA and is part of the Institute for Complex Engineered Systems at Carnegie Mellon University.

9. REFERENCES

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

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

[3] Benchmarq Microelectronics, Inc. *1998 Data Book*. 1998.

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

[5] M. Doyle and J. Newman, "The Use of Mathematical Modeling in the Design of Lithium/Polymer Battery Systems," Electrochimica Acta, vol. 40, no. 13-14; 1995; pp. 2191-2196.

[6] M. Doyle, J. Newman, and J. Reimers, "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.

[7] S. Graham, P. Kessler, M. McKusick, "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.

[8] K. Govil, E. Chan, and H. Wasserman, "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.

[9] T. Kuroda, K. Suzuki, S. Mita, T. Fujita, F. Yamane, F. Sano, A. Chiba, Y. Watanabe, K. Matsuda, T. Maeda, T. Sakurai, and T. Furuyama, "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.

[10] D. Linden, *Handbook of Batteries*, 2nd ed. New York: McGraw-Hill, Inc. 1995.

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

[12] PC Magazine, "Travelling powerhouses," January 19, 1999.

[13] T. Pering and R. Brodersen, "Energy efficient voltage scheduling for real-time operating systems," Fourth IEEE Real-Time Technology and Applications Symposium, Works-In-Progress Session, June, 1998.

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

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

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

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

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

[19] N. Weste, K. Eshraghian, *Principles of CMOS VLSI Design: A Systems Perspective*, 2nd ed. Reading, MA: Addison-Wesley Publishing Company, 1993.

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