The Benefits of Event–Driven Energy Accounting in Power-Sensitive Systems

Frank Bellosa
Department of Computer Science (Operating Systems), University of Erlangen,
Martensstr. 1, 91058 Erlangen, Germany
bellosa@acm.org

A prerequisite of energy-aware scheduling is precise knowledge of any activity inside the computer system. Embedded hardware monitors (e.g., processor performance counters) have proved to offer valuable information in the field of performance analysis. The same approach can be applied to investigate the energy usage patterns of individual threads. We use information about active hardware units (e.g., integer/floating-point unit, cache/memory interface) gathered by event counters to establish a thread-specific energy accounting. The evaluation shows that the correlation of events and energy values provides the necessary information for energy-aware scheduling policies.

Our approach to OS-directed power management adds the energy usage pattern to the runtime context of a thread. Depending on the field of application we present two scenarios that benefit from applying energy usage patterns: Workstations with passive cooling on the one hand and battery-powered mobile systems on the other hand.
Energy-aware scheduling evaluates the energy usage of each thread and throttles the system activity so that the scheduling goal is achieved. In workstations we throttle the system if the average energy use exceeds a predefined power-dissipation capacity. This makes a compact, noiseless and affordable system design possible that meets sporadic yet high demands in computing power. Nowadays, more and more mobile systems offer the features of reducible clock speed and dynamic voltage scaling. Energy-aware scheduling can employ these features to yield a longer battery life by slowing down low-priority threads while preserving a certain quality of service.

1 Introduction

OS-directed halting of the CPU for short periods of time can smooth the energy consumption of a workstation to keep the average power below the limit that is imposed by the thermal conductivity of the device. This makes a compact and affordable design without noisy fans feasible that meets sporadic yet high demands in computing power. Todays operating systems stop the processor only in the idle thread. However, this policy does not throttle the energy consumption in the case of high load. Our approach enforces additional halt-cycles to defer the use of energy and therefore to limit the average power in the case of high load.

A genuine saving in energy is possible with processors that offer the feature of tunable clock speed to reduce the power of the processor. Some of today's embedded processors (AMD ELAN SC400 [1], StrongARM1100 [16], Hitachi SuperSH [12]) already exhibit this feature. Scaling the core supply voltage according to the clock frequency offers further opportunities for an energy-efficient operation of a device [11, 13, 20, 24, 27]. Particularly the academic results concerning variable voltage design have been applied to latest products (Intel Pentium Mobile III [17], Transmeta Crusoe [26]).

A prerequisite for any kind of decisions made by the system software is precise information about relevant events. Practically all information which is significant to CPU scheduling is gathered on the basis of threads. If system software wants to benefit from dynamic system throttling, dynamic speed setting and dynamic voltage scaling, it has to know the energy-usage patterns of individual threads. Without thread-specific energy accounting, serious power management is not possible within current operating systems, in which adjusting power-management parameters is at best a black art.

This paper discusses the fine-grained and thread-specific online analysis of energy-usage patterns that are fed back into the scheduler to control the CPU clock speed. Because the expected power savings are based on very subtle effects (battery discharge analysis, speed-voltage correlation, clock gating of unused processor components like the floating-point unit or the bus interface) a trace-driven simulation is unfeasible. Therefore we favor an evaluation with real hardware. The first results of an implementation are presented.

Section 2 describes the energy-related behavior of the hardware and motivates a thread-specific energy accounting. The *Joule Watcher* exploiting the information provided by event counters is presented in Section 3. In Section 4 we propose some on-line scheduling policies that throttle the CPU to reduce the energy consumption, and we round up with a conclusion in Section 5.

2 Energy-Related Characteristics of System Components

The typical components of computer systems that consume a significant fraction of energy are the processor, the memory, the display and the I/O system (e.g., graphics card, network adapter, disks). In mobile systems the properties of batteries have to be considered.
Beside the option of zoned backlighting [10] there is no way to software-control energy savings of the display in an active system. The energy consumption of the I/O system covered by dynamic power-down management [6, 14, 18, 19] is out of the scope of this paper. Here we focus on the CPU- and memory-related aspects of power management.

The battery, the processor and the memory remain as the energy critical components under investigation.

2.1 Battery

The capacity of a battery is dominated by two factors: the load power and the intermittence of discharge.

The charge capacity is the total amount of energy a battery can deliver when discharged at a constant current, called a 1C discharge rate, over a a defined period (normally 1 hour) [22]. The discharge rate has a non-linear impact on the total amount of power a battery can deliver. A typical lithium-ion battery has the following characteristics [15]:

Discharge Rate	Usable Battery Capacity (Normalized to 1C Discharge Rate)
C/5	107%
C/2	104%
1C	100%
2C	94%
4C	86%

A high discharge rate (e.g., >4C) can lower the usable battery capacity to 70%-80%. Additionally the usable battery capacity can be shortened by an intermittent load. Duty cycles of 25% can reduce the usable capacity by 40% compared to a continuous load with the same average power [22].

Due to the influence of discharge rate and load intermittence on the battery capacity, our energy accounting should consider both effects. Furthermore the scheduler should throttle the system activity in a way that establishes continuous load of a modest level. Sporadic high loads should be avoided.

2.2 Processor

Several factors influence the energy consumption of a CPU:

- Each functional unit working on behalf of an instruction needs a certain amount of energy, e.g., the instruction fetch/ decode unit, the pipeline stages of the ALU, the floating point unit need a certain amount of energy to perform an operation. Depending on the instruction mix the electrical power usage of a CPU can vary. By considering the properties of a CPU, the compiler could generate code that is optimized for minimal energy usage [25]. The energy accounting should cope with a variable energy usage without specific instrumentation or algorithmic knowledge of the application.
- The CPU speed and a dynamically adapted supply voltage change the energy performance [24, 27]. Neglecting the impact of external effects on CPU performance (e.g., memory latencies, bus contention) the energy per operation remains constant when reducing the frequency at a fixed voltage. However, the energy per operation is proportional to s^2, if frequency and voltage is changed by a factor s within the allowable limits of operation.
  Therefore a reduced clock speed at a low supply voltage should be targeted by advanced scheduling policies. The energy accounting should be aware of electrical characteristics of a CPU driven at different operation points.

2.3 Memory

The performance of a processor depends on the fast supply of data and code. Consequently an improvement in cache efficiency leads to an efficient use of all processor units and therefore to an improvement in energy efficiency. This benefit to energy efficiency is reinforced because each main memory access and each bus transaction needs energy that is consumed by the drivers of the bus system and the memory itself [7, 21]. According to our measurements (see section 3) the data exchange between processor and second level cache that is placed outside the CPU core significantly contributes to the energy consumption of the system.
In addition to cache- and memory-aware code optimizations for saving energy, scheduling strategies can respect the cache affinity of individual threads [4] and improve the cache reuse of threads that use shared memory segments [3] in order to avoid bus transactions and CPU stall cycles due to cache misses. Consequently those memory-conscious scheduling strategies alleviate the power consumption of a system as well.

2.4 Requirements for Energy Profiling

Energy profiling should provide all relevant information so that an energy-aware scheduler can tune the timing, sequence and speed of execution. Halting the CPU is the simplest form of speed adjustment that only offers two speed levels.
Tuning the clock speed presumes an entity to which a specific clock speed must be assigned and a policy to determine the appropriate speed settings. Previous work in the field of dynamic speed settings [13, 24, 27] focus on adjusting the clock speed in intervals. This approach is motivated by deadline driven real-time strategies with a predefined workload. We envision interactive devices running unknown and maybe dynamically loaded software. Therefore our approach has to support any threads even though their application specific properties are not known in advance.

A first approach is the measurement and classification of applications and parts of applications according to energy-usage patterns [5, 9]. On-line measurements employing a digital multimeter [10] have to correlate data that was gathered by an energy monitor with profiling data sampled by the target system. Beside the additional hardware effort for the multimeter, the software overhead for sampling (which has negative effects on the cache performance in embedded systems with small caches), and the overhead to correlate the data, an instruction-accurate power analysis is nearly impossible due to the inertia of the voltage regulator associated to the processor and the inertia of the multimeter's measurement unit.
A novel approach to on-line energy profiling should consider the power related-effects of each instruction without significant influence on the execution of the target system.

3 *Joule Watcher* Energy Accounting

Our approach to on-line energy accounting uses counters embedded in the target hardware to register events that imply the consumption of a certain amount of energy. The number and type of these events is stored and updated in the thread and system context. This information allows us to easily estimate energy consumption of individual threads and the whole system. Because the counters register events without interfering with the application, the only overhead is caused by saving and restoring counter values in the context switching routine.

We will show that counter values strongly correlate to a specific energy consumption, so we have found a cheap and easy methodology for power consumption monitoring.

3.1 Measurement Methodology

Our initial target environment is a simple Pentium II 350 PC running a Linux 2.2.14 operating system (The next system under observation will be an embedded PC using an AMD ELAN processor [1] that offers variable clock speed). Context switch routines and kernel data structures are modified to hold the values of the two available performance-monitoring event counters. These counters are realized in the P6 family as registers and can be configured to count one of several events. The accumulated counter values can be accessed though the /proc-file system.

Calibration software has to find the correlation of events and energy values. Therefore synthetic micro-benchmarks trigger events of a certain type and frequency for several seconds, while a simple multimeter with integrated power monitor measures the electrical power of the whole system. Because of the 1 Watt resolution of our multimeter, we smooth the power value by calculating the average of 5 consecutive readings of the meter. A direct measurement of the processor's and the chip set's current was not possible due to the complex power supply of the target system.

Each point in the figures represents a particular power measurement with the corresponding number of events. We tune the parameters of the calibrations software to come close to a certain number of events. But the precise number of events vacillates between measurements. Therefore we get more a cloud of reading points than multiple power measurements of a certain deviation for predefined numbers of events.

3.2 Measurements

In the first measurement, a single micro-benchmark triggering integer operations is running with variable sleep interval. Thus a variant number of instructions per second is executed. Figure 1 demonstrates the correlation of the number of micro operations per second executed by the Pentium II CPU with the energy per second consumed by the total system.

We see a nearly linear increase of energy with a rising number of integer and control-flow operations which entail micro operations (UOPS). Our synthetic calibration software runs totally in the registers and in the first level cache, without any main memory or second-level cache references, and without any TLB-misses. We conclude that we can assign an energy value of $1.83 \cdot 10^{-8}$ Ws (Watt x Second) to each retired microoperation.

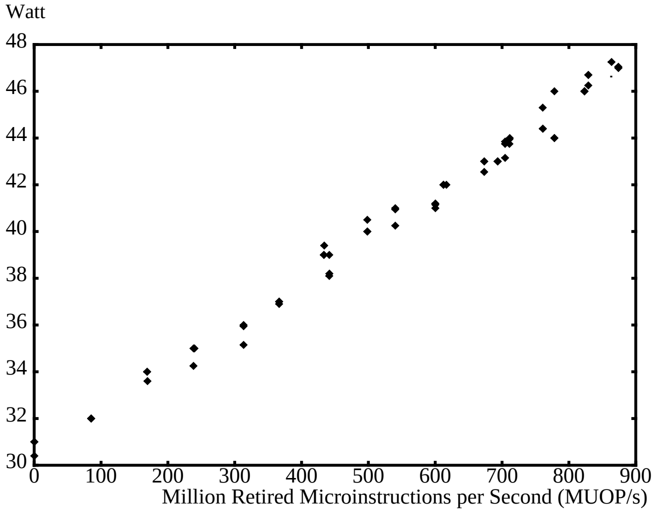

FIG. 1. Correlation of retired microinstructions and energy consumption

A functional unit of the CPU responsible for additional energy consumption is the floating-point unit. Because it is hard to build a calibration software that exclusively triggers floating point operations, we timely interleave two micro-benchmarks *I* and *F* with variable sleep intervals. Benchmark *I* issues integer operations whereas micro-benchmark *F* prevailingly issues floating point operations but also a few integer operations for loops and blocking routines. The calibration software counts the total number of operations and the number of floating point operations while varying the ratio between the two event types. Having already calculated the energy per integer micro operation, we can subtract the energy for the integer operations at each measurement point from the total energy gathered from the multimeter. Figure 2 shows the energy consumption of the system running a certain amount of floating point operations.

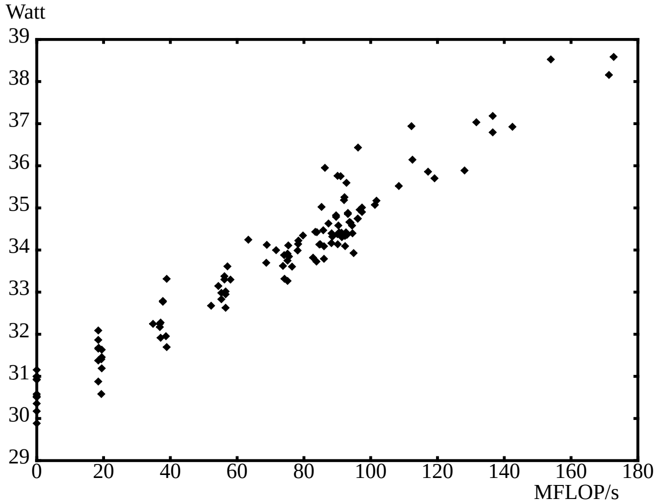

FIG. 2. Correlation of floating point operations and energy consumption

We can see a linear correlation with a 1 Watt variation. The reason is the coarse 1 Watt resolution of our multimeter Exploiting this measurement, we get an energy value of $4.29 \cdot 10^{-8}$ Ws per floating point operation.

The next candidate for energy consumption is the second level cache.

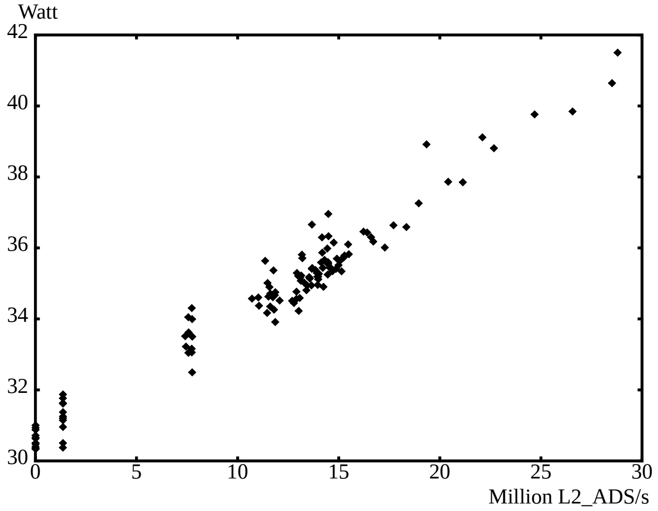

FIG. 3. Correlation of L2 Cache references and energy consumption

Similar to the floating-point calibration software we vary the ratio between integer and memory access operations which result in first level cache misses handled by the second level cache.
The plot (see Figure 3) shows a linear correlation between the number of second level cache address strobes (L2_ADS) and the energy consumption. We get an energy value of $3.56 \cdot 10^{-7}$ Ws per second level cache reference.

The last measurement deals with memory requests as a consequence of cache misses. Our calibration environment mixes load/store and integer operations. The load/store operations may cause second level cache misses which trigger bus transactions and memory chip activity.

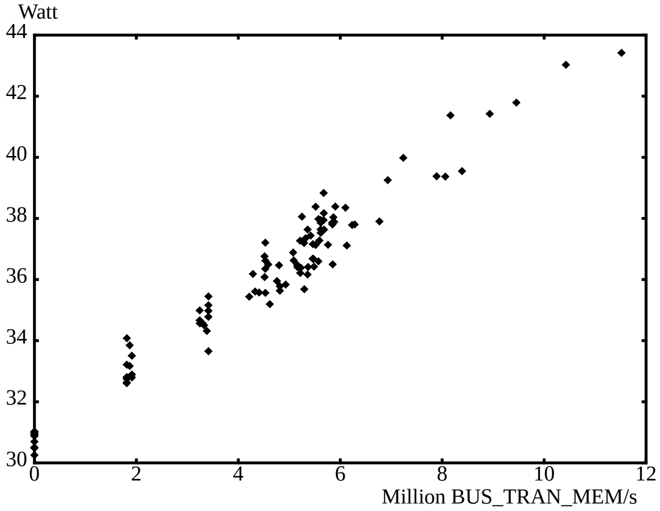

FIG. 4. Correlation of main memory references and energy consumption

Depending on the CPU's ability to tolerate several outstanding transactions the CPU stalls while waiting for data which is not found in the cache. Looking at the relationship between energy and memory references, we can see that each memory reference accounts approximately for an energy value of $1.14 \cdot 10^{-6}$ Ws.

3.3 Discussion

Event counters have proved to provide detailed information about the use of hardware units. Furthermore, the hardware events can be mapped to energy amounts of fixed size. This mapping has to be evaluated in an initial calibration procedure for each type of system. But once we have energy values for each event, the energy accounting is very precise without additional overhead. Because the counters were designed for performance monitoring, they have no immediate relation to energy consumption. Nevertheless do we get results within the resolution of our external multimeter for both, synthetic and real-world applications (e.g., ghostscript, gimp,...) by exploiting the four types of events presented in this paper (microoperation, floating-point operations, second-level address strobes and memory transactions).

Todays architectures only offer a limited number of counters. The Pentium P6 family used in our initial implementation comprises two counters that can be mapped to a variety of events. But we cannot monitor four events simultaneously. Therefore we have to multiplex the counters while a thread is running. To compare Joule Watcher with power measurements for real-world applications we had to re-configure the counters to register all necessary types of events while certain application procedures were repeated.
The technique of sampling and multiplexing is accepted in the field of performance monitoring [2] where a well-known application is analyzed in several runs. Our goal is to monitor any type of applications. This includes single run applications as well as just-in-time compiled applications. Because we cannot guarantee an exact energy measurement of those applications we assess sampling as a critical technique when deployed in the field of energy accounting.

It is evident that the energy per event is not constant if the clock-speed varies. Therefore the calibration software has to gauge various speed levels for each type of CPU event.

Performance-monitoring counters have become a common tool in the field of performance profiling. With the upcoming of energy accounting systems we expect that future hardware generations will provide a rich set of event counters which are fully dedicated to energy-related activities.

4 Energy-Aware Scheduling

To demonstrate the benefits of an event-triggered energy accounting, we propose energy-aware scheduling strategies that throttle the average power in workstations and that reduce the energy consumption in mobile systems.

4.1 Throttling of Average System Power

Many users of workstations ask for high performance systems at affordable costs. Additionally they favor compact package designs and noiseless systems that do not disturb the working environment. The latter requests can be satisfied with passive cooling. But low-power components that operate continuously with passive cooling offer the desired performance at a high price, that is normally justified only for mobile systems.

Typically the high computing power of personal workstations is used at short bursts. This sporadic demand for high power motivates our approach to power throttling. We employ affordable standard components that normally rely on active cooling like fans. But we throttle the system activity if the average energy use exceeds the predefined power-dissipation capacity of the passive cooling.
The *Joule Watcher* energy accounting offers information about the energy use with regard to individual threads as well as the whole system. If the average system-energy approximates to the threshold that guarantees a safe operation, a dedicated *throttling thread* that halts the CPU (e.g., by executing the hlt-instruction) is activated. The throttling thread is enqueued in the run-queues of the scheduler so that high priority and highly interactive threads still can run, but low-priority threads consuming a lot of energy become throttled until the system has cooled down. Special care has to be taken for the display server (e.g., the X window system) that should be boosted to a priority level above the throttling thread as long as a safe operation can be guaranteed. By throttling system-local applications the display server will be throttled in its activity as well. However remote applications can impose high load on the display server so that the server has to be throttled as a last resort.

4.2 Reduction of Energy Consumption

The operating system has various options to improve the energy efficiency of battery-powered mobile systems.

- If the operating system monitors a high number of cache misses after restarting a thread (compulsory misses) it should extend the time-slice of the thread to reduce the number of restarts. Additionally, scheduling strategies can respect the cache affinity of individual threads [4, 28] and improve the cache reuse of threads that use shared memory segments [3] in order to avoid bus transactions and CPU stall cycles due to cache misses. In addition to the improvement in performance all those memory-conscious scheduling strategies alleviate the power consumption of a system.
- A high number of main memory references (e.g., bus transactions related to main memory) and a low number of instructions indicates that the speed of execution is dominated by the main memory latency. Without noticeable performance degradation the scheduler can improve the energy efficiency by throttling the clock speed of the processor working on behalf of a latency-bound application. By this means the clock frequency is kept at such a low level that applications run close to their optimal operation point.
  Application measurements have demonstrated [21] that this operation point is application dependant.
- Analogously to the policy of average-power throttling (see Section 4.1) we can throttle threads equivalently to their priority to save energy and to yield a longer battery life. The lower the priority the lower is the speed. Once again special care has to be taken concerning the display server and other timing-sensitive threads. In the field of soft real-time applications (e.g., movie or audio players) we have to investigate energy-aware APIs, and admission and control policies for power management under real-time constraints (see subsection 4.3).

4.3 Energy-Aware API

There is a great demand for an API to empower applications to affect their energy usage [8, 23] as soon as the energy usage patterns contribute to the runtime context of a thread and as soon as the power becomes a first-class operating system managed resource. We envision two types of self-tuning applications:

- According to the amount of energy granted by the operating system the applications can tune their quality of service, e.g., the resolution of images, the sampling rate of voice or the frequency of updates is adapted so that the requested system lifetime is achieved.
  This scenario requires power admission and control policies in the operating system and an OS-interface that supports requests for energy.
- Many applications can tune their algorithmic behavior towards a better energy efficiency, e.g., depending on the characteristics of a device (energy costs for CPU instructions, memory access, cache size,...) the application can find the best trade-off between main memory access and data de-/compression in the cache. This scenario requires an interface to feed the application with information of the energy-related costs and the on-line measurements of the applications induced system activities.

5 Conclusion

The more the operating system knows what is going on inside the hardware the more it can adapt the execution of threads to the needs of the user. With the upcoming of power-sensitive devices, the operating system scheduler has to move from a CPU-centric approach to activity control of all power related components. Thread specific and event-driven energy accounting is an absolute prerequisite to reach this goal.

Our approach to event-driven energy accounting has proved to offer thread-specific energy patterns without any overhead. The current implementation can only use a small number of counters that were intended originally for performance profiling. If the operating system technology is ready to deal with a variety of counters in all locations of the hardware it is just a small step to embed new counters which are exclusively devoted to energy accounting.

We expect thread-specific speed settings in combination with event driven energy accounting to become an unrenounceable element of future operating systems for power-sensitive devices.

6 References

[1] AMD. *ElanSC400 and ElanSC410 Microcontroller User's Manual*, 1997.

[2] Anderson, J., Berc, L., Dean, J., Ghemawat, S., Henzinger, M., Leung, S.-T., Sites, R., Vandervoorde, M., Waldspurger, C., and Weihl, W. Continuous profiling: Where have all the cycles gone? In *Proceedings of the 16th Symposium on Operating Systems Principles SOSP'97* (Oct 1997).

[3] Bellosa, F. Follow-on scheduling: Using tlb information to reduce cache misses. In *Proceedings of the 16th Symposium on Operating Systems Principles SOSP'97, Work in Progress Session* (Oct 1997).

[4] Bellosa, F., and Steckermeier, M. The performance implications of locality information usage in shared-memory multiprocessors. *Journal of Parallel and Distributed Computing 37*, 1 (Aug. 1996), 1–2.

[5] Benini, L., Bogliolo, A., Cavallucci, S., and Ricco, B. Monitoring system activity of os-directed dynamic power managament. In *Proceedings of the International Symposium on Low-Power Electronics and Design ISLPED'98* (1998).

[6] Benini, L., Bogliolo, A., and De Micheli, G. Dynamic power management of electronic systems. In *Proceedings of the International Conference on Computer-Aided Design ICCAD'98* (1998).

[7] Benini, L., Macii, A., Macii, E., and Poncino, M. Selective instruction compression for memory energy reduction in embedded systems. In *Proceedings of the International Symposium on Low-Power Electronics and Design ISLPED'99* (Aug 1999).

[8] C.Ellis, Lebeck, A., and Vahdat, A. System support for energy management in mobile and embedded workloads: A white paper. Tech. rep., Duke University, Department of Computer Science, Oct 1999.

[9] Ellis, C. The case for higher level power management. In *Proceedings of HotOS'99* (Mar 1999).

[10] Flinn, J., and Satyanarayanan, M. Energy-aware adaption for mobile applications. In *Proceedings of the 17th Symposium on Operating Systems Principles SOSP'99* (Dec 1999).

[11] Govil, K., Chan, E., and Wassermann, H. Comparing algorithms for dynamic speed-setting of a low-power cpu. In *Proceedings of the 1st Conference on Mobile Computing and Networking MOBICOM'95* (Mar 1995). also as technical report TR-95-017, ICSI Berkeley, Apr. 1995.

[12] Hitachi. *SuperH (SH) 32-Bit RISC MCU/MPU Series SH7750 Hardware Manual*, Jul 1998.

[13] Hong, I., Potkonjak, M., and Srivastava, M. On-line scheduling of hard real-time tasks on variable voltage processor. In *Proceedings of the International Conference on Computer-Aided Design ICCAD'98* (Nov 1998).

[14] IBM. Adaptive power management for mobile hard drives. White Paper, Jan 99.

[15] Intel. *Mobile Power Guidelines 2000 Rev 1.0*, Dec 1998.

[16] Intel. *Intel StrongARM SA-1100 Microprocessor Developer's Manual*, Apr 1999.

[17] Intel. *Intel SpeedStep Technology*, Jan 2000.

[18] Intel, and anf Toshiba, M. *Advanced Configuration and Power Interface Specification 1.0b*, Feb 1999.

[19] Lorch, J., and Smith, A. Software strategies for portable computer energy management. *IEEE Personal Communications Magazine 5*, 3 (June 1998), 60–73.

[20] Martin, T., and Siewiorek, D. A power metric for mobile systems. In *Proceedings of the International Symposium on Low-Power Electronics and Design ISLPED'96* (1996).

[21] Martin, T., and Siewiorek, D. The impact of battery capacity and memory bandwidth on cpu speed-setting: a case study. In *Proceedings of the International Symposium on Low-Power Electronics and Design ISLPED'99* (Aug 1999).

[22] Martin, T. L. *Balancing Batteries, Power and Performance: System Issues in CPU Speed-Setting for Mobile Computing*. PhD thesis, Department of Electrical and Computer Engineering, Carnegie Mellon University, 1999.

[23] Noble, B. System support for mobile, adaptive applications. *IEEE Personal Communications 7*, 1 (Feb. 2000), 44–49.

[24] Pering, T., and Broderson, R. Energy efficient voltage scheduling for real-time operating systems. In *Proceedings of the 4th IEEE Real-Time Technology and Applications Symposium RTAS'98, Work in Progress Session* (Jun 1998).

[25] Simunic, T., Benini, L., and De Micheli, G. Energy-efficient design of battery-powered embedded systems. In *Proceedings of the International Symposium on Low-Power Electronics and Design ISLPED'98* (Jun 1998).

[26] Transmeta. *The Technology behind Crusoe Processors*, Jan 2000.

[27] Weiser, M., Welch, B., Demers, A., and Shenker, S. Scheduling for reduced cpu energy. In *Proceedings of the First Symposium on Operating System Design and Implementation OSDI'94* (Nov 1994).

[28] Weissman, B. Performance counters and state sharing annotations: a unified approach to thread locality. In *Proceedings of the Eighth International Conference on Architectural Support for Programming Languages and Operating Systems ASPLOS'98* (Oct 1998).