Power Management and Dynamic Voltage Scaling: Myths and Facts

David Snowdon, Sergio Ruocco and Gernot Heiser

National ICT Australia*
and
School of Computer Science and Engineering
University of NSW, Sydney 2052, Australia
Firstname.Lastname@nicta.com.au

Abstract

This paper investigates the validity of common approaches to power management based on dynamic voltage scaling (DVS). Using instrumented hardware and appropriate operating-system support, we account separately for energy consumed by the processor and the memory system. We find that memory often contributes significantly to overall power consumption, which leads to a much more complex relationship between energy consumption and core voltage and frequency than is frequently assumed. As a consequence, we find that the voltage and frequency setting that minimises energy consumption is dependent on system characteristics, and, more importantly, on the application-specific balance of memory and CPU activity. The optimal setting of core voltage and frequency therefore requires either a-priori analysis of the application or, where this is not feasible, power monitoring at run time.

1 Introduction

Dynamic voltage scaling (DVS) is a standard technique for managing the power consumption of a system [22]. It is based on the fact that the *dynamic* (switching) power P of CMOS circuits is strongly dependent on the core voltage V and the clock frequency f according to

$$P \propto fV^2. \tag{1}$$

Under the assumption that the number of clock cycles required for a computation is independent of the core frequency, the execution time is inversely proportional to the frequency. The total energy E for the computation is then proportional to the square of the voltage:

$$E \propto V^2. \tag{2}$$

Note that the total energy for a computation does in this simple model not depend on the frequency, but a reduced core voltage requires a reduction of the clock frequency and therefore implies a longer overall execution time.

The assumptions behind Eqn. 2 are highly dubious, as they ignore other system components, in particular the front-side bus and memory [2]. Those other components impact the execution time of a program, leading to a much more complex dependence on the processor frequency. Furthermore, those components themselves consume energy, and that energy consumption scales differently than the processor's. While memory power may be dominated by CPU power in high-end systems, this is not the case for embedded systems using low-power processors. Finally, Eqn. 1 is not even necessarily a good model of the power consumption of a modern processor, as for modern CMOS circuits the *static* energy consumption can no longer be ignored.

This paper presents a measurement-based examination of the effect of DVS on the energy required to execute applications on a modern embedded system. We independently measure processor and memory power consumption on a representative platform, and find that the behaviour is quite different from what is expected by the simple model. As a consequence, we find that more sophisticated methods are required in order to manage limited energy resources well.

2 Related Work

There exists a large body of work on both dynamic and static voltage scaling [3, 5, 6, 13, 14, 18, 22]. Many of the

* National ICT Australia is funded through the Australian Government's *Backing Australia's Ability* initiative, in part through the Australian Research Council.

ideas developed by Weiser et al. [22] and Govil et al. [5] form the basis of these algorithms: that the CPU idle (slack) time should be minimised by slowing the CPU core frequency. This reduces the DVS problem to estimating the idle time.

Other studies have examined frequency and voltage scaling in time-sensitive systems [11, 19, 20, 24]. One approach is to use timing information which is available in real-time systems. This can allow static schedules to be developed such that processor utilisation is maximised (all deadlines are only just met). Modifications can be made to the schedule on-line in order to make use of slack-time made available by processes which complete before their deadlines.

Weissel and Bellosa [23] measured the effect of frequency scaling on the performance and total power consumption of an XScale-based computer running several benchmarks. They examined the number of memory references and instructions executed at runtime in order to determine the memory dependence of an application, and thus estimate its response to a reduction in CPU frequency (a memory-bound application will be limited by memory speed rather than CPU speed). They determine what CPU core frequency will result in a 10% or less reduction in performance for the process. No voltage scaling was used in this work. Choi et al [10] refined this work to allow a dynamic rather than static tradeoff between power and performance reduction by characterising process memory and CPU usage at run-time.

There are several previous studies of the power consumed by real computers. The most relevant is that of Miyoshi et al [16] who examine a set of microbenchmarks running on two different platforms. They find that in some cases the lowest-performance setting may not give the lowest total energy. They also provide a methodology for choosing which settings shoud be ignored.

Fan et al. [2] used a modified simulator to estimate the power consumed by an XScale-based device with power-aware SDRAM. They observe that, owing to a system's static power consumption (particularly owing to DRAM), the energy reduction via frequency scaling can be outweighed by the energy resulting from a longer execution time. Their results indicate that an aggressive memory power-down policy such as that which they had previously developed [1] can reduce this effect.

Martin [14] studied the effect of frequency scaling on battery lifetime, developing a system for identifying the CPU frequency at which the most computation could be performed using a single battery charge.

Flinn et al. [4] conducted a similar study of the ItSY pocket computer, using external power management and off-line evaluation. Micro-benchmarks were used to study the effect of frequency scaling on the processor's performance and power consumption. Voltage scaling was not examined.

3 Benchmarks

A number of benchmarks were used to represent typical workloads for a variety of embedded system. The majority of these were taken from the MiBench [7] suite, along with four others, also representing typical embedded applications, described in previous work [21]. Each benchmark in this collection represents a fixed amount of "work" for the system, therefore the total energy for each benchmark is directly comparable.

MiBench is a suite developed by the academic community with the explicit aim of representing embedded workloads. The particular benchmarks used were selected based on their resource requirements: many of the MiBench tests require large input data which could not be accommodated on the RAM disk of our disk-less system. The future addition of network and disk support to PLEB 2 should allow the full suite to be executed. Furthermore, benchmarks which ran for less than four seconds were excluded to avoid measuring start-up and wind-down energy.

All output was discarded to avoid filesystem overheads and resource constraints.

4 Experimental Platform

The experiments were performed on PLEB 2 [21], a power-aware computer based on ARM XScale processor running a standard Linux OS, augmented with current sensors to measure the power consumption of the CPU core, RAM, and I/O devices.

4.1 Hardware

PLEB 2 is a single-board computer based on the Intel XScale PXA255 [9]. The PXA255 was chosen as being representative of high-performance, low-power CPUs designed for use in embedded systems. It consists of a 400MHz ARMv5TE-compatible core combined with a set of on-chip peripheral units including memory, interrupt, DMA and LCD controllers.

The computer consists of the CPU, SDRAM and flash memory. The SDRAM is implemented using two Micron MT48LC16M16A2 ICs [15], and the flash is implemented using two Intel TE28F320 ICs [8]. Three switching power supplies generate core, memory and IO power. A minimal set of peripherals (infra-red, USB, and serial port) are provided on-board. An 8-bit microcontroller performs a supervisory role. The PXA255, flash, SDRAM and the power supply represent the core of a typical embedded system.

Linux 2.4.19, Linux 2.6.8, L4ka::Pistachio [12], and Iguana [17] have been adapted to run on PLEB 2 hardware.

4.2 Power management features

PLEB 2 supports a number of power management features. Frequency/voltage scaling and low-power modes are software-managed throttling mechanisms of interest.

The PXA255 supports the frequency scaling of three main clocks:

- the CPU core (*core*);
- the PXBus (*pxbus*): an internal bus that links the CPU core, DMA/Bridge, memory controller and LCD controller;
- the memory clock (*memclk*): drives the memory and LCD controller.

While the hardware which controls the frequency settings will allow a large number of combinations of core, pxbus and memclk frequencies, only a subset of these will allow the system to operate correctly (i.e. within the upper and lower frequency limits for all components in the system). All of the valid setpoints are shown in in Table 1.

The power-supply chip used in PLEB 2 (Epson S1F81100) supports voltage scaling. The core (CPU) voltage can be varied in 0.1V increments. This voltage is set to the appropriate value as given in the PXA255 developer's manual [9].

The PXA255, SDRAM and flash memory all support low-power states. In these states, the devices have a reduced functionality, but use significantly less power. For the PXA255, there are several modes: run/turbo, idle, 33MHz idle and sleep. Run/turbo are active modes where the CPU is running. Turbo mode is a mechanism for performing fast frequency changes by synchronously switching a clock divider. Idle mode stops the CPU core clock but does not halt its generation, avoiding loss of state and supporting a fast recovery to run mode. 33MHz idle and sleep mode are progressively deeper sleep states that require longer recovery times.

The Micron SDRAM also supports low-power modes. While not being accessed, it maintains an active standby mode which, according to the datasheet [15], consumes a maximum of 132mW per chip (although the typical idle current has been measured to be much lower on PLEB 2). If the chip is put into power-down mode (data is retained, but the chip must be refreshed periodically) it consumes a maximum of 6.6mW.

The Intel Flash chips have very effective automatic power management: according to the datasheet [8] they use less than 1mW unless being read/written, and even less when in one of the available power-down mode. Since the power consumption of flash is very small compared to CPU and memory we ignore it in our discussion.

4.3 Measurement system

The computing core of PLEB 2 is supplied by three power supplies. These are dedicated to the CPU core (nominally 1.5V), memory bus devices (3.3V), and IO devices (3.3V). Each of these power supplies is instrumented using a small series resistor and an amplifier. The analogue-to-digital converter within the on-board microcontroller reads the resulting signal. The voltage on the supplies is assumed to be sufficiently constant as to not require measurement. The power can then be calculated via $P = IV$.

Data collected is transferred from the microcontroller to the PXA255 via an I2C bus. Statistical sampling is used to associate the power readings with the running processes. For each sample, a series of interrupts are generated to record which process the sample should be attributed to, and to start the transfer of data. The resulting information is made available at user level at run-time.

The overhead associated with handling the measurements on the PXA255 will introduce an error to the measurement of time, cache misses, writebacks, etc. This is because the CPU will spend some time handling the interrupts, and because the events associated with the interrupt handlers. This overhead varies between 2% and 10% of execution time depending on the nature of the application. Because the sampling rate is independent of the processor speed the overheads will vary. The measured power is not affected by the measurement system because the power samples are always taken when running the real system.

Further information regarding the measurement system, its overheads and validation is given in a previous publication [21].

Cache miss and write-back numbers were determined using the PXA255's performance monitoring unit and appropriate OS support.

All experiments were run on Linux 2.4.19, modified to provide memory and CPU energy accounting through the `/proc` file system, from where it can be accessed by a modified `time()` function.

5 Methodology

A number of operating points were chosen. Each operating point defines a specific hardware configuration under

V_{core} (V)	f_{cpu} (MHz)	f_{pxbus} (MHz)	f_{mem} (MHz)
1.0	99.5	50.0	99.5
1.0	199.1	99.5	99.5
1.1	298.6	99.5	99.5
1.3	398.1	99.5	99.5
1.3	398.1	199.5	99.5

Table 1. Hardware configurations under test

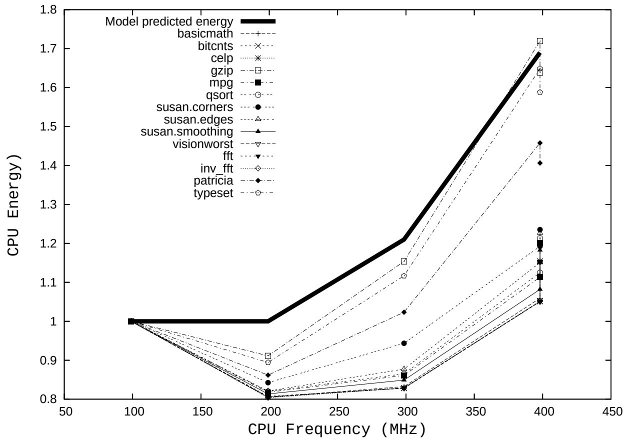

Figure 1. Normalised CPU energy

test. Five operating points (as shown in Table 1) were initially available as defined by the frequency-setting code in Linux — those where operating points where the memory frequency was 99.5MHz. By varying (and even overclocking) the memory frequency it would be possible obtain a larger number of operating points than those used.

The platform was configured according to each of the operating points and the benchmarks executed. The mean current was measured for the CPU and memory power supplies (since no devices are connected to the IO supply, that supply was deemed irrelevant) and recorded on the RAM disk. The results were later transferred to a PC for analysis. Each experiment was repeated 10 times and the results averaged, standard deviations were less than 1%.

6 Results

Fig. 1 shows the processor's energy consumption of the various benchmarks as a function of the core frequency, normalised to the energy consumption at the lowest clock rate (100MHz). There are two values at the highest frequency (400MHz) correspond with the two different processor bus frequencies used at that setting.

The bold line shows the prediction of Eqn. 2. Contrary to naïve theory, we find that the energy is in fact quite dependent on the clock rate: Increasing the core frequency from 100MHz to 200MHz (at unchanged core voltage) results in a drop of total energy for all benchmarks. Further frequency increases lead to increases in energy, as they are accompanied by voltage scaling. The benchmarks consistently stay *below* the predictions of Eqn. 2 — in other words, Eqn. 2 *over-estimates* the benefit from DVS.

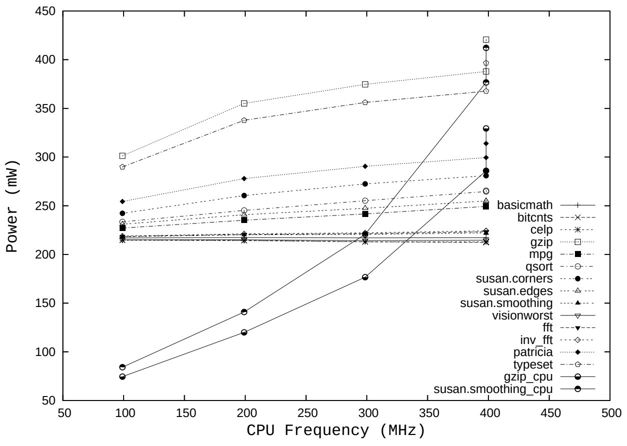

Figure 2. Memory power

This effect can be explained with the processor's *static* power consumption, which is independent of the core frequency. As the runs with a faster clock require less total time, the total static energy consumption is less in those cases.

The two benchmarks whose energy, under frequency *and* voltage scaling (from 200 to 400MHz), grows steeper than the theory are `gzip` and `typeset`, which are both memory-limited while the others are CPU-limited. Memory is always clocked at the same rate, and therefore a memory-limited application's execution time benefits less from an increase in the core frequency than CPU-limited applications. Hence, for those benchmarks the influence of the static energy increases with increasing clock speed.

Memory behaves differently than the processor, and is best examined in the power, rather than the energy dimension. As Fig. 2 shows, most benchmarks consume very little memory power, and it is very weakly dependent on the core frequency. These benchmarks run essentially out of cache and cause very little memory traffic, so we see mainly the static energy of RAM, which is attributed to DRAM refreshes, leakage and powering input/output buffers. The memory-intensive benchmarks show a strong frequency dependence at lower clock rates, which then flattens out, a consequence of the saturation of the memory system in

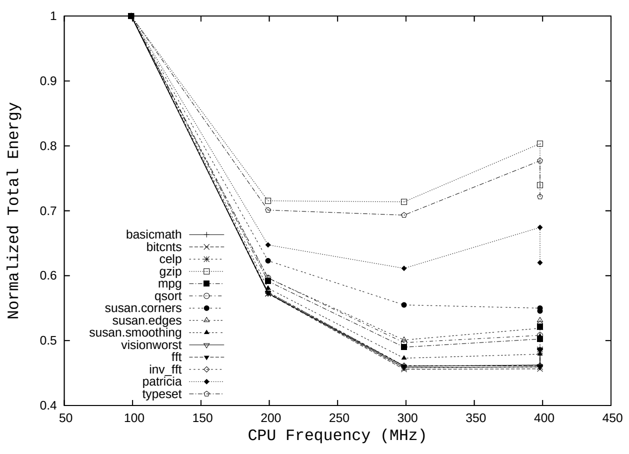

Figure 3. Normalised total energy

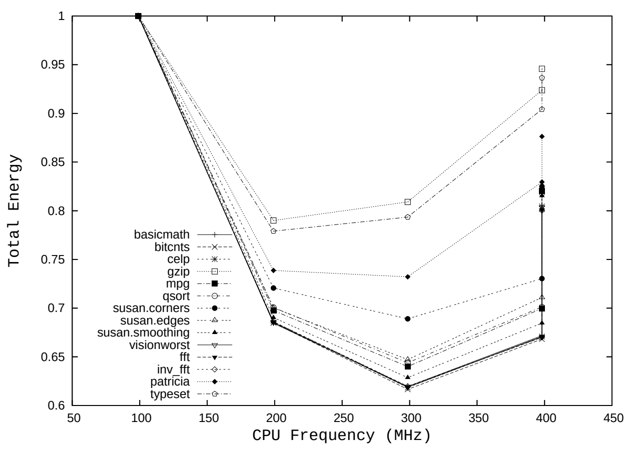

Figure 4. Normalised total energy padded to equal total time

those runs. Note also that the difference of the two data points at the highest frequency (corresponding to different bus frequencies, see Table 1) is highest for the memory-limited benchmarks.

The flat curves mean that memory power scales very little with core frequency. Translated into total energy (accounting for total execution time) this means that memory energy use is actually *lowest* at the *highest* clock rates.

Also shown in Fig. 2 (strongly rising curves) is the minimal and maximal CPU power consumed by the benchmarks. It can be seen that, compared to memory power, the range is relatively small, and except at the highest core frequency, CPU power is dominated by memory power.

This helps explain Fig. 3, which shows the total (CPU+memory) energy for the execution of the benchmarks. Inclusion of the memory energy leads to results which bear no similarity whatsoever with the model of Eqn. 2, and, contrary to folklore, shows that the highest clock rates actually *minimise* the energy requirements of the computations!

This result is somewhat misleading, however. The higher clock rates lead to faster completions of the runs. A more fair comparison of energy requirements needs to compare the energy used over the same total time period [2]. This means that the slack time remaining after an early completion results in an idle system, which still consumes power. We assume that the system, when idle, switches to the low-power idle mode from which it can be woken up quickly when an interrupt arrives (indicating a new computation task).

The result is shown in Fig. 4. We can clearly see that for all benchmarks the total energy is minimised at some intermediate frequency, neither the highest nor the lowest.

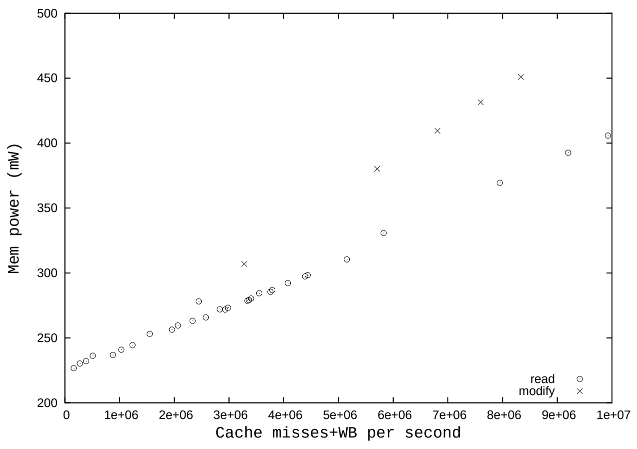

Figure 5. Memory power as a function of the cache miss rate

That frequency depends on the particular benchmark, it is lower for the memory-intensive than for the CPU-intensive benchmarks. The optimal frequency will obviously also depend on characteristics of the system, such as type and size of memory.

Weissel and Bellosa [23] model memory energy by using a performance counter to measure memory references, and assume that the energy cost of each memory reference is the same. Many systems (such as ours) do not provide such performance counters. One can attempt to approximate the number of memory references by the number of cache misses (for which the PXA255 has performance counters). Fig. 5 shows that this is inaccurate, as a single cache miss can produce either one or two memory references. For this figure we ran synthetic benchmarks which in a tight loop contained load instructions (*read* case in the figure), or load instructions followed immediately by a store to the same memory location (*modify*). Varying numbers of nop instructions were inserted to vary the cache miss rate. We see that the modify case has a higher memory-energy cost per cache miss than the read, owing to the higher number of memory operations (write-back followed by a refill, compared to just the refill). A realistic load would lie somewhere in between those extremes, but it would be difficult to predict where. In addition, the presence of read and write buffers significantly complicates any modelling of memory traffic from cache miss rates.

7 Discussion

Our results show that traditional model of Eqn. 2 is not suitable for estimating the effect of DVS on modern processors, as it ignores the effect of static power, and grossly distorts reality. Furthermore, our measurements confirm that memory contributes significantly to the power consumption of embedded systems, and attempts to manage power without taking memory into account will likely lead to incorrect results.

Static power is also important for memory, and should be ideally be minimised by keeping as much RAM as possible in a low-power state. Furthermore, modelling dynamic memory power by measuring cache misses can produce misleading results, unless read and write misses can be measured separately (and even then it would be difficult to achieve good accuracy), owing to the complex memory-access patterns resulting from a processor which augments caches by read and write buffers.

Overall we find that the dependence of the energy cost of a computation on the processor core voltage and frequency is a complex function of system configuration and properties of the application, too complex to predict an energy-optimal operating point for DVS using simple models.

In some cases, the optimal operation may be determined by off-line measurements, but in general this is only possible if application loads are known well in advance. The only alternative is to determine the optimal voltage and frequency setting at run-time, based on the observation of the actual power consumption.

While we have shown how, with the help of some relatively simple instrumentation, such observation can be performed on off-the-shelf processors, this only provides the input data for successful power management. The required algorithms and policies remain the subject of future work.

References

[1] X. Fan, C. Ellis, and A. Lebeck. Memory controller policies for DRAM power management. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)*, 2001.

[2] X. Fan, C. Ellis, and A. Lebeck. Synergy between power-aware memory and processor voltage scaling. In *Proceedings of the Proceedings of the Workshop on Power Aware Computer Systems (PACS)*. Springer-Verlag, December 2003.

[3] K. Flautner, S. K. Reinhardt, and T. N. Mudge. Automatic performance setting for dynamic voltage scaling. In *Mobile Computing and Networking*, pages 260–271, 2001.

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

[5] K. Govil, E. Chan, and H. Wasserman. Comparing algorithm for dynamic speed-setting of a low-power CPU. In *Mobile Computing and Networking*, pages 13–25, 1995.

[6] D. Grunwald, P. Levis, K. I. Farkas, C. B. Morrey III, and M. Neufeld. Policies for dynamic clock scheduling. In *Proceedings of the 4th Symposium on Operating Systems Design and Implementation (OSDI)*, pages 73–86, 2000.

[7] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown. MiBench: A free, commercially representative embedded benchmark suite. In *IEEE 4th Annual Workshop on Workload Characterization*, December 2001.

[8] *Intel Advanced+ Boot Block Flash Memory (C3)*. http://www.intel.com/design/flcomp/products/b3_c3/techdocs.htm, May 2005.

[9] Intel PXA255 processor developer's manual. http://www.intel.com/design/pca/products/pxa255/techdocs.htm, 2005.

[10] R. S. Kihwan Choi and M. Pedram. Fine-grained dynamic voltage and frequency scaling for precise energy and performance trade-off based on the ratio of off-chip access to on-chip computation times. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, January 2005.

[11] W. Kim, D. Shin, H. Yun, J. Kim, and S. Min. Performance comparison of dynamic voltage scaling algorithms for hard real-time systems. In *Proceedings of the Symposium on Real-Time and Embedded Technology and Applications*, 2002.

[12] L4Ka Team. L4Ka::Pistachio kernel. http://l4ka.org/projects/pistachio/.

[13] J. R. Lorch and A. J. Smith. Improving dynamic voltage scaling algorithms with PACE. In *SIGMETRICS/Performance*, pages 50–61, 2001.

[14] T. L. Martin. *Balancing Batteries, Power, and Performance: System Issues in CPU Speed-Setting for Mobile Computing*. PhD thesis, Carnegie Mellon University, Pittsburgh, Pennsylvania, 2001.

[15] Micron Technology Inc. *Micron Synchronous DRAM Datasheet*, 2003.

[16] A. Miyoshi, C. Lefurgy, E. C. Hensbergen, R. Rajamony, and R. Rajkumar. Critical power slope: understanding the

runtime effects of frequency scaling. In *Proceedings of the 16th international conference on Supercomputing*, pages 35 – 44, 2002.

[17] National ICT Australia, http://ertos.nicta.com.au/Research/ESF/Iguana. *Iguana Embedded Operating System*.

[18] T. Pering, T. Burd, and R. Brodersen. The simulation and evaluation of dynamic voltage scaling algorithms. In *Proceedings of the International Symposium on Low Power electronics and Design*, pages 76–81, 1998.

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

[20] D. Shin, J. Kim, and S. Lee. Low-energy intra-task voltage scheduling using static timing analysis. In *Proceedings of Design Automation Conference*, pages 438–443, 2001.

[21] D. C. Snowdon, S. M. Petters, and G. Heiser. Power measurement as the basis for power management. In *2005 WS Operat. System Platforms for Embedded Real-Time applications*, Palma, Mallorca, Jul 2005.

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

[23] A. Weissel and F. Bellosa. Process cruise control: event-driven clock scaling for dynamic power management. In *Proceedings of the international conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES02)*, 2002.

[24] W. Yuan and K. Nahrstedt. Energy-efficient soft real-time CPU scheduling for mobile multimedia systems. In *Proceedings of the nineteenth ACM symposium on operating systems principles*, pages 149–163, Bolton Landing, NY, USA, 2003. ACM Press.