Accurate On-line Prediction of Processor and Memory Energy Usage Under Voltage Scaling

David C. Snowdon
NICTA
University of New South Wales
Sydney, Australia

Stefan M. Petters
NICTA
University of New South Wales
Sydney, Australia

Gernot Heiser
NICTA
University of New South Wales
Open Kernel Labs
Sydney, Australia

First.Last@nicta.com.au

ABSTRACT

Minimising energy use is an important factor in the operation of many classes of embedded systems — in particular, battery-powered devices. Dynamic voltage and frequency scaling (DVFS) provides some control over a processor's performance and energy consumption. In order to employ DVFS for managing a system's energy use, it is necessary to predict the effect this scaling has on the system's total energy consumption. Simple (yet widely-used) energy models lead to dramatically incorrect results for important classes of application programs.

Predicting the energy used under scaling requires (i) a prediction of the dependency of program performance (and hence execution time) on the frequencies and (ii) a prediction of the power drawn by the execution as a function of the frequencies and voltages.

As both of these characteristics are workload-specific our approach builds a model that, given a workload execution at one frequency setpoint, will predict the run-time and power at any other frequency setpoint. We assume temporal locality (which is valid for the vast majority of applications) so predicting the characteristics of one time slice, frame, or other instance of a task, will imply the characteristics of subsequent time slices, frames or instances (e.g. MPEG video decoding).

We present a systematic approach to building these models for a hardware platform, determining the best performance counters and weights. This characterisation, done once for a particular platform, produces platform-specific but workload-independent performance and power models.

We implemented the model on a real system and evaluated it under a comprehensive benchmark suite against measurements of the actual energy consumption. The results show that the model can accurately predict the energy use of a wide class of applications and is highly responsive to changes in the application behaviour.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Modeling techniques; D.4.8 [**Operating Systems**]: Performance—*Modeling and prediction*

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
EMSOFT'07, September 30–October 3, 2007, Salzburg, Austria.
Copyright 2007 ACM 978-1-59593-825-1/07/0009 ...$5.00.

General Terms

Management, Performance

Keywords

Energy, Power, DVS, DVFS, PMC, Performance counter

1. INTRODUCTION

Power consumption is a critical factor in many embedded devices, particularly battery-powered mobile systems, but is also receiving increasing attention in server systems as a means to cut costs, prevent overheating and manage air-conditioning load.

Modern processors typically support dynamic voltage and frequency scaling (DVFS) as a means to manage their power consumption. This is based on the fact that reducing the operating frequency of a CMOS circuit allows a reduction in the supply voltage for that circuit, resulting in a lower energy consumed per cycle. This is usually expressed as

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

where V is the circuit voltage and f the processor's clock rate. The intention is that a system's CPU performance be reduced in order to obtain a corresponding reduction in power.

While the mechanism exists for throttling a processor's speed, when to switch and which speed to choose in order to achieve a goal (such as minimising average power) is, and remains, a difficult problem.

We postulate that effective frequency scaling, truly trading quality of service against energy consumption, requires the following:

1. an accurate prediction of the performance impact of a frequency change
2. an accurate prediction of the power drawn while running the applications at the changed frequency and voltage settings;
3. knowledge of the costs associated with switching frequencies.

In other words, in order to effectively make performance vs. energy decisions, it is necessary to know how much performance will be lost, and how much energy will be saved. However, much of the related work in frequency and voltage scheduling assumes simplified models for both the performance (postulate 1) and power (postulate 2) while ignoring switching overheads. These models have been shown to lead to an *increase* in power consumption [13], or missed power saving opportunities [18].

Predicting either power or performance is made more difficult by application specificity. Workloads' change in power and performance when changing frequencies is different depending on the exact instruction mix, memory and IO actions, etc. Applications can be pre-characterised for the platform, but this does not allow for dynamic applications or dynamic input data. We are motivated by modern mobile embedded systems such as mobile phones or PDAs. Such systems support the execution of downloaded code, making it infeasible to characterise the behaviour of all application programs *a priori*, yet require efficient energy management without human direction. Similarly, these systems deal with dynamic input data such as MPEG video streams where the data itself has a significant effect on the workload characteristics. Therefore some method of characterising the workload must be arrived at in order to understand the relationship between the frequencies, energy and performance.

Recently we developed a new solution to the first postulate, by developing a highly-accurate model that uses performance counters to predict, at run time, the performance of virtually arbitrary workloads under frequency scaling [22]. Our approach builds a model that, given a workload execution at one frequency setpoint, will predict the run-time and power at any other frequency setpoint. Assuming temporal locality (which is valid for the vast majority of applications) means the characteristics of one time slice, frame, or other instance of a task, will be similar to the characteristics of subsequent time slices, frames or instances in most cases. Typical examples are MPEG video decoding, interrupt handlers, and decompression. With these models we can therefore reason about the future behaviour of a workload when frequency scaling (as shown via the implementation in our previous work) by observing the workload currently running.

System power has often been calculated using Equation 1, and more recently, using a static power component. However, doing so makes a number of assumptions. Firstly it assumes that the same number of transistors switch on each cycle, which is rarely the case for a complex microprocessor. Similarly, it assumes that every part of the system is clocked at the same frequency or fraction of that frequency. Again, this is not the case in modern systems where there may be many other frequencies, such as those used for driving the system bus, memory, and peripherals. These clock frequencies are typically one of several possible fractions of the main core clock (i.e. not linearly dependent).

Lastly, it does not consider memory and IO device power, which may be dynamic, depending on the running software. We note that devices' time spent in their active, high-power, mode is sometimes dependent on the CPU performance (i.e. a shorter workload execution time may relate to a device spending a smaller amount of time in its run-mode, saving energy). We do not address the effect of idle modes or variation in IO device power in this paper (other than including it as a part of the static power), but do note that predicting workload run-time is required in order to calculate a system's energy use.

This paper extends our previous work in performance prediction, arriving at a generalised model for the prediction of the system power and energy under scaling of CPU, bus and memory frequencies for dynamic workloads at run-time. The methodology for building this model could be applied to an arbitrary platform with any number of frequencies, although the accuracy of the model is limited by the performance instrumentation available. This paper therefore addresses postulate 2, above. This leaves as future work the evaluation and avoidance of frequency switching overheads (postulate 3), focussing, for the present, on the modelling issues.

The contributions of this paper are:

- an examination of the energy behaviour of a low-power embedded platform;
- the extension of performance-counter-based power *estimation* models to the accurate *prediction* of power consumption;
- a generalised power and energy model for frequency and voltage scaling;
- an instantiation of that model for an actual hardware platform;
- a rigorous evaluation of the model based on measurements of actual power consumption.

In the remainder of the paper we will first discuss related work. We will then discuss the unsuitability of naive approaches to DVFS-based power management and the need for accurate models that predict a system's energy response to frequency scaling. We will then summarise our previous work on the accurate prediction of *performance* under DVFS. This work is then extended, becoming a model allowing prediction of the power and energy response. We will outline the calibration and evaluation of the new model, and discuss the its accuracy on a real system with real-life benchmarks. The paper concludes with a discussion of future work.

2. RELATED WORK

DVFS was first discussed as an energy-saving technique by Weiser et al. [24]. They used *millions of instructions per joule* (MIPJ) as a metric when evaluating their scheme, however they ignored static power (which was not unreasonable at the time). Since then, a large body of work has accumulated based on the idea that a lower core frequency and thus longer execution time results in lower energy consumption.

Grunwald et al [8] first published an empirical evaluation of DVFS algorithms, concluding that the then-developed algorithms did not save significant amounts of energy.

One clear reason for this is that the algorithms attempted to minimise idle time, ignoring the effect of static power. Reducing the CPU frequency increases execution time, leading to a decrease in the amount of time spent idle. If the energy benefits of running at the lower frequency do not offset the energy spent due to the reduced time in the idle mode (because of the static power), the total used is *increased*. These effects were discussed at length by Miyoshi et al [13]. Of interest in this context is that this Miyoshi's work assumed a very simple execution time model.

Martin and Siewiorek [11] discussed system-level speed setting policies in order to maximise the number of operations per battery discharge in a mobile computer. They discuss the two big assumptions made by the body of previous work at the time: that performance is proportional to clock frequency, and that power is proportional to fCV^2. Their most significant contribution was the observation that a workload's execution time is not necessarily linearly related to the CPU clock frequency. They also suggested the need for a more complete power model. This was supported by measurements on a real system, however, only a single application has been measured. While they suggest that this non-ideal relationship between frequency and performance needs to be addressed, they make only a passing reference to the need for an application specific model.

These observations were discussed and utilised in a paper by Seth et al [18]. The authors concluded that taking advantage of the non-linear relationship between CPU frequency and performance

could yield significant extra energy savings when frequency scaling.

DVFS work designed to minimise performance loss, such as that of Weissel and Bellosa [25] implicitly avoids issues with the overall system power consumption because the approach explicitly aims to keep changes in overall execution time small.

Acquaviva et al [1] describe a speed setting policy which saves energy on a fixed voltage system because of the non-ideal performance vs. frequency characteristics caused by memory accesses. The authors pre-characterise a particular application (an MP3 decoder) and verify their conclusions by comparing the results from a simulator, but no experiments on real hardware were performed. Also Acquaviva et al do not consider the effect of static power consumption which ultimately lead to their approach of minimising idel times rather than using available sleep modes.

This paper presents methods for building both performance and power models for a system. There is various work using performance counters to estimate the power used by a processor, but to our knowledge this model is the first to be used for predicting the power when frequency scaling (i.e. using frequencies and voltages as part of the model).

Bellosa [2] demonstrated that it was, in principle, possible to correlate hardware performance-monitoring counter (PMC) readings with the energy consumption of a CPU. His students Waitz [23] and Fruth [7] used this approach to implement energy accounting in Linux on an Intel PXA255 processor. Fruth worked around that processor's limitation of only two PMCs by switching PMC settings every 10 ms.

Bircher et al. [3] used the PMC-based approach on the Pentium 4, which has many counters, and investigated the suitability of different counters. Research using the IBM Power processors [5] developed a predictor of program phases in terms of power consumption, similar to dynamic branch predictors. This was later ported to the Intel PXA 255 processor [4]. Peddersen and Parameswaran [16] investigated custom CPU designs which provided PMCs counting events specifically selected for estimating the power consumption of the CPU.

There have been attempts at using hardware sensors to directly measure the power used in a system. A framework based on external multimeters and a dedicated profiling platform [6] can be used to characterise the workload of a closed system, but does not provide feedback to the system under test. We refined this approach by implementing on-board power-measurement support that allowed us to account for CPU, memory and I/O power separately [20]. While this approach supports low-overhead run-time measurements (and thus avoids the need for *a-priori* workload or platform characterisation), the need for extra hardware makes it unsuitable for off-the-shelf devices, and is a deterrent to manufacturers. Furthermore, without a model for the system behaviour, it is impossible to predict the power under conditions other than those being measured.

Similar work has been published regarding performance models. However, as discussed in our previous work [22], these have a large performance overhead, and are generally highly platform specific (compared with our methodology which has a small overhead and could be deployed on any platform providing PMCs).

In the area of I/O modelling and energy minimisation, Nathuji et al. [14] investigated the scheduling of I/O activity on an Intel PXA255 to increase device idle times. They implemented this scheme in the Linux kernel, supported by compiler annotations of the code. The approach did not make any use of online HW information. Weissel et al [26] examined the effect of deferring I/O operations on the effective use of sleep states and found significant energy savings.

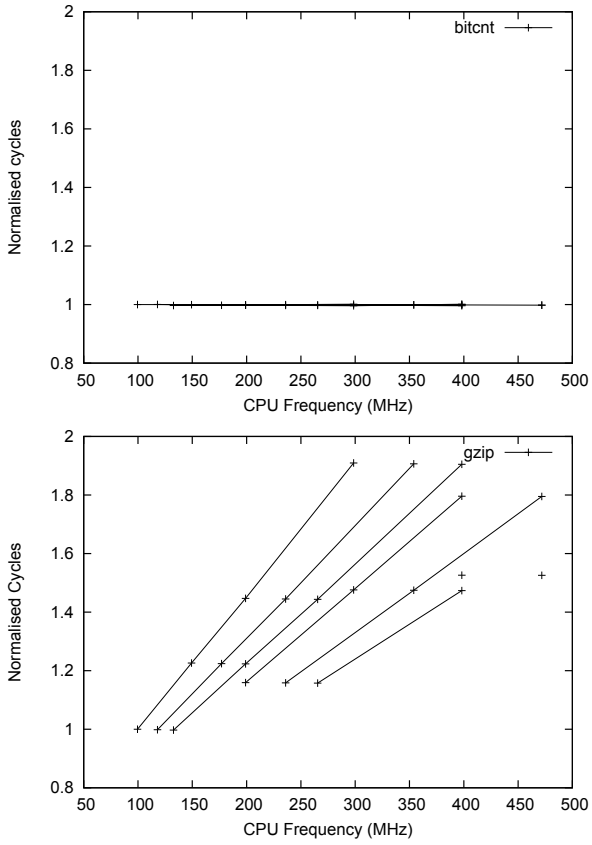

Figure 1: Normalised cycle count for a CPU bound (bitcnt) and a memory bound (gzip) workload. Lines join points of equal bus and memory frequency.

Lastly, the realisation that energy use is important in embedded systems has lead to work that attempts to integrate it with overall operating-system resource management. Examples are Nemesis [15] and the *currentcy* model of EcoSystem [27].

3. MOTIVATION: PERFORMANCE VS. ENERGY

As mentioned, previous work has shown that models which were commonly used for designing and implementing DVFS schemes are flawed. In the belief that the lowest frequency leads to the highest energy efficiency (Equation 1), the simplest schemes attempt to run the processor as slowly as possible for given quality of service constraints. This can counterintuitively lead to lower energy efficiency than consistently running at the highest possible frequency.

Firstly we examined the execution-time issues discussed by Martin and Siewiorek [11]. In Figure 1 we have depicted two extreme applications used in our evaluation in Section 6. We varied three clock frequencies in the XScale processor: the CPU core, the internal bus, and the memory bus clock. Figure 1 shows the relative number of cycles of the core clock for different frequency setting for the CPU core, internal bus, and memory bus clocks. For illustration purposes the cycles have been normalised to the slowest frequency setpoint for each example. Setpoints having the same

clock frequency respectively for internal bus and memory bus have been connected by lines.

The CPU bound benchmark has a nearly constant number of cycles to execute and thus behaves ideally as with the simple models (i.e. the performance is proportional to the CPU clock period). However, the memory bound benchmark shows that not only does the number of cycles grow at higher CPU frequencies (i.e. a sub-linear speed up), there is a dependency on the internal and memory bus frequencies, which leads to close to a factor of two variation in the number of cycles.

Alternately, we could consider that, when we reduce the CPU frequency, the performance impact is less than what would be expected if we didn't consider the memory effects. This leads to better energy saving opportunities as discussed by Seth et al [18].

The power used by the system is also dependent on more than the processor performing computations. This is modelled in recent literature via a static component to the power model. The static power consumption (P_{static}) of embedded systems has moved now into the same order of magnitude as the dynamic power consumption (P_{dyn}) by the processor and in cases even surpasses it.

Ultimately we are interested in the amount of energy consumed by an application, which is dependent on the static and dynamic power consumption as well as the execution time (T). This is depicted here as

$$E = P_{static}T + \int_0^T P_{dyn}dt \tag{2}$$

In general terms P_{dyn} rises with higher clock frequencies, however, in terms of energy this is partially compensated for by a shorter execution time. The static power is constant and thus the static energy consumed is a linear function of the execution time. The energy function in Equation 2 is highly sensitive to the execution time as well as the balance of dynamic and static power in the system. The latter includes the static power drawn by memory and I/O devices, and as such the details of this balance are highly platform-specific.

To further illustrate the dependence of the behaviour of total energy on workload characteristics, we ran two synthetic benchmarks: CPUBOUND performs no memory accesses after a short warmup phase, while *readbound* executes almost exclusively LOAD instructions. Figure 2 shows total energy as a function of application *performance*, normalised to the highest frequency. Here, performance at frequency f is expressed as the ratio of execution times at f_{max} and f. It therefore represents the true trade off between execution time and energy. We again varied bus and memory frequencies and lines connect the data points where those are equal.

It is obvious from the graphs that the energy use of the CPU-bound process is completely controlled by the CPU frequency (although not at all in the way expected by the naive model of Equation 1) while this frequency has a minimal effect on the memory-bound benchmark.

These figures lead us to agree with the previous work suggesting that conventional DVFS theory has little to do with reality and in fact breaks down catastrophically in the sense that it can predict the opposite of what truly happens. It is clear from the above that much more sophisticated execution time and energy models are required.

4. TIME MODEL

Our previous work [22] developed an accurate model for execution time under DVFS, which we summarise here for completeness.

The execution time, T, for a given piece of software can be represented as a sum of the time spent waiting for each component in the system. This time is, in turn, proportional to the inverse of the clock frequency for that component:

$$T = \frac{C_{cpu}}{f_{cpu}} + \frac{C_{bus}}{f_{bus}} + \frac{C_{mem}}{f_{mem}} + \frac{C_{io}}{f_{io}} + \dots \tag{3}$$

The coefficients C_x are workload-specific, and can be thought of as the number cycles (of the component's clock, f_x) the rest of the system is waiting for component x. In this paper we do not consider I/O operations' effect on performance, and power drawn by I/O peripherals is lumped into static power. An effective model for the power consumed by I/O devices is the focus of work yet to be published.

Performance counters can be used to estimate C_{bus} and C_{mem}. This is because the performance counters available in most modern CPUs count events which relate to the memory performance (such as stall cycles and cache misses). For an arbitrary set of performance counters, we represent the bus and memory coefficients as a linear combination of their readings,

$$\begin{aligned} C_{bus} &= \alpha_1 PMC_1 + \alpha_2 PMC_2 + \dots \\ C_{mem} &= \beta_1 PMC_1 + \beta_2 PMC_2 + \dots \end{aligned} \tag{4}$$

where the constants α_i and β_i depend on the platform's characteristics, but not on the workload. Hence it is possible to *characterise the platform* once and then use the results for all workloads. With a separate measurement of execution time (typically

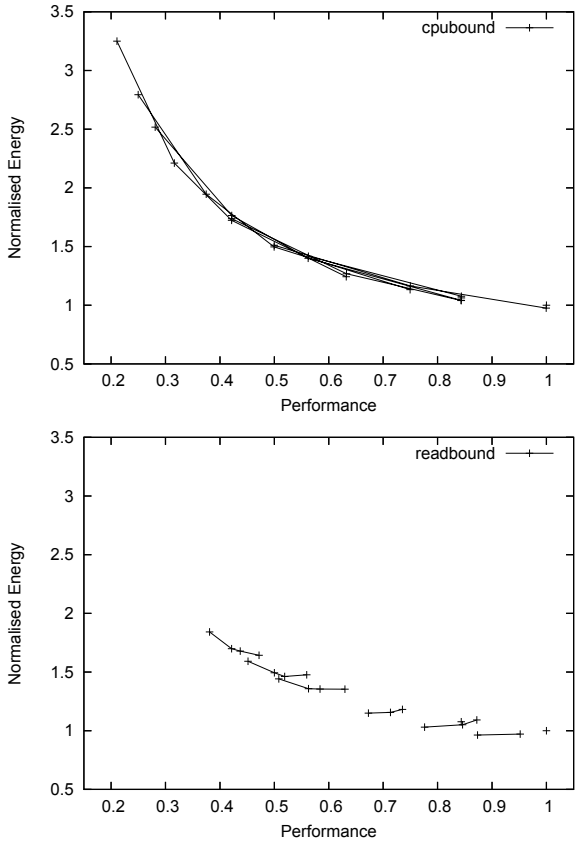

Figure 2: Normalised Energy vs. Performance for `cpubound` and `readbound` on PLEB 2. Lines join points of equal bus and memory frequency.

using the processor's cycle counter), Equation 3 can be rewritten as

$$C_{cpu} = C_{tot} - \frac{f_{cpu}}{f_{bus}} C_{bus} - \frac{f_{cpu}}{f_{mem}} C_{mem} \tag{5}$$

leading to the ability to calculate all of the constants in Equation 3.

This model describes the frequency-dependence of the execution time of software. We can use this to predict the execution time at arbitrary frequency settings f'_x from observations at frequency settings f_x:

$$C'_{tot} = C_{tot} - \frac{f_{cpu}}{f_{bus}} C_{bus} - \frac{f_{cpu}}{f_{mem}} C_{mem} + \frac{f'_{cpu}}{f'_{bus}} C_{bus} + \frac{f'_{cpu}}{f'_{mem}} C_{mem} \tag{6}$$

The relative performance s under frequency scaling from f to f' is

$$s = \frac{f_{cpu}}{f'_{cpu}} \times \frac{C'_{tot}}{C_{tot}} \tag{7}$$

where C'_{tot} is obtained from Equation 6.

We characterised a representative embedded platform and found the predictions of Equation 6 to be highly accurate [22].

In the same work, the models were then used to implement a frequency scaling scheme in order to demonstrate the validity of using the model for one time slice to predict the characteristics of the same software in the next. The linux scheduler was modified such that on each time-slice interrupt, the performance counters were measured and the models were used to predict the performance at every frequency setpoint. The frequency setpoint predicted to have the closest performance to the target was chosen. End-to-end measurements revealed that the overall performance was very close to the target performance (with most of the error being caused by the discrete frequency setpoints). This shows that, for the large number of benchmark applications tested, the models can be used to predict the workload's future performance.

5. ENERGY MODEL

We can use the same line of argument to predict the power drawn for the execution of a program under DVFS. For the time being we are making a number of assumptions, some of which will be reviewed in the future.

Firstly, managing I/O power is beyond the scope of this paper. Hence we assume that I/O power is unaffected by DVFS, and thus can be treated as static power. This assumption is accurate for programs that do not perform any (or very little) physical I/O and the OS consequently keeps all I/O peripherals in the same power state throughout the program's execution.

Secondly, we consider leakage power as part of the system's static power. This is not necessarily true, as the processor's capacitances may not get fully charged at the highest clock rates. However, we could not detect any frequency-dependence of static power on the processor we used, so this assumption seems justified.

Under these assumptions Equation 1 holds for *dynamic* power, and the *dynamic* energy consumed during a time interval Δt is

$$E_{dyn} \propto fV^2 \Delta t. \tag{8}$$

If the time interval is expressed in CPU cycles, $cyc = f\Delta t$, this becomes

$$E_{dyn} \propto cyc \times V^2. \tag{9}$$

This energy corresponds to the energy $E = \frac{1}{2}CV^2$ of a capacitor C, it represents the energy used to charge and discharge the circuit's capacitances (such as the gate capacitance on a transistor) during each clock cycle.

However, we cannot assume this dynamic energy to be independent of workload properties, as the number of transistors switched on each clock cycle depends in general on the executing program. Instead we can use *event counters* to capture the effects of program behaviour.

Modern processors have performance counters which can be used to count a number of different types of events, some of which are correlated with power [2,25]. The main question is whether the countable events provide enough information to predict power.

We can try to model the energy consumed during a time interval Δt as a linear combination of the various system frequencies and m available event counts:

$$\begin{aligned} E = {} & V_{cpu}^2(\gamma_1 f_{cpu} + \gamma_2 f_{bus} + \gamma_3 f_{mem})\Delta t + \\ & V_{cpu}^2(\alpha_0 PMC_0 + \cdots + \alpha_m PMC_m) + \\ & \gamma_4 f_{mem}\Delta t + \beta_0 PMC_0 + \cdots + \beta_m PMC_m + \\ & P_{static}\Delta t, \end{aligned} \tag{10}$$

where PMC_i is the event count of performance monitor i during the interval Δt, and α_i, β_i and γ_i are the coefficients of the model. Note that f_{mem} occurs twice, once with a V^2 factor and once without. The reason is that this frequency represents the memory bus, which interfaces with the on-chip memory controller that is voltage scaled as well as the memory chips which run at constant voltage.

The rate r_i of the event measured by counter i is $r_i = \frac{PMC_i}{\Delta t}$, which lets us express power as

$$\begin{aligned} P = {} & V_{cpu}^2(\gamma_1 f_{cpu} + \gamma_2 f_{bus} + \gamma_3 f_{mem}) + \\ & V_{cpu}^2(\alpha_0 r_0 + \cdots + \alpha_m r_m) + \\ & \gamma_4 f_{mem} + \beta_0 r_0 + \cdots + \beta_m r_m + P_{static} \end{aligned} \tag{11}$$

This equation can be used to predict the power at one frequency setting from measurements obtained at a different setting, similar to Equation 6. However, the execution-time model is also required in this process, in order to predict the changes in the performance-counter rates. The mathematics is straightforward but the resulting formulas are unwieldy, which is why we do not present them here explicitly.

Provided that the PMCs cover enough of the relevant events, Equation 11 should be able to predict the scaled power. Combined with the model of execution time of Section 4, we should then be able to predict energy usage under DVFS. The important point is that the coefficients α_i, β_i and γ_i should be independent of workload characteristics and only depend on the hardware. They can therefore be determined once for each platform. This can be done by running a representative set of *calibration* programs on the target system, measuring all events and the energy required to execute them. To maximise the amount of information gathered, the voltage should be varied to all allowable settings for each frequency.

While modern processors support counting a reasonably large number of different events (e.g. 14 on the XScale PXA255) the number of events that can be counted concurrently tends to be much smaller (only 2 on the PXA255). In practice we therefore need a further step: determining the most relevant events to count.

We do this by performing an exhaustive search of all n-parameter models. Each n-parameter model is fit to the calibration data using least squares regression. The model with the highest R^2 was chosen as the best model for n. While we were able to perform this operation in reasonable time for up to 50 parameters, well known statistical methods exist for narrowing the search space for larger numbers. The parameters thus selected provide the best model

for the calibration workloads used and the performance counters available. Provided there exists a suitable set of performance counters, and the calibration workload was representative enough, this method will find a power model that can be used to manage energy consumption.

6. EVALUATION

When instantiating and evaluating the models on actual hardware, we used end-to-end measurements of energy, performance counters and time for real-world benchmark suites at various frequencies on a typical embedded platform. We chose end-to-end benchmarks over measurements of periodic slices of a workload in order to provide a constant, predictable workload at varying frequencies. Here we describe our target hardware platform (PLEB 2), our energy-measurement methodology and the benchmarks used.

6.1 Platform

PLEB 2 is a single-board computer, consisting of an PXA255 processor [10] which, as specified, can be clocked at up to 400MHz (but was over-clocked to 471MHz for the purposes of collecting data for this paper). It has 64MB SDRAM and 8MB Flash. The PXA255 is based on an ARMv5T-compatible XScale core with a 7-stage integer and 8-stage memory pipeline. It has split L1 caches and TLBs, write, fill and pend buffers. The data cache supports a write-back policy.

The system includes a LAN91C111 network interface IC, which was used to transfer benchmark software and results. The interface was disabled during all measurements to avoid interrupts modifying the workload, but it was not placed in a low-power mode. There are several other components on-board which are all considered to use a constant power. These, combined with the CPU, memory and flash, result in a significant constant static power.

Various voltages are generated by an S1F81100 power-management chip from a constant 5V source. It consists of three buck converters which switch to the CPU core, memory and IO interface voltages. The maximum efficiency for the core voltage is quoted at 80%, with 90% for the memory and IO interface voltages. The S1F81100 technical manual [17] gives graphs showing approximately constant efficiency for the loads of interest. This chip can be controlled by the PXA255 via an I2C bus, allowing for adjustment of the processor's core voltage. All other circuits run at a fixed 3.3V. The S1F81100 is supplied with a constant 5V from a generic laboratory power supply via the energy measurement circuit outlined in Section 6.2. Further details of the PLEB 2 design are available [19].

A number of frequencies are generated by the PXA255's clock management unit. These include the core clock (f_{cpu}), system bus clock (f_{bus}), SDRAM clock (f_{mem}), peripheral bus clock (f_{io}), real-time clock (f_{rtc}), etc. Because of the way these clocks are synthesised, only certain combinations can be generated. We call each of these clock combinations a *setpoint*. Typical of a real system, only f_{cpu}, f_{bus} and f_{mem} are varied in these experiments.

All possible combinations of f_{cpu}, f_{bus} and f_{mem} were considered, including those outside the chip's specifications. The frequencies were limited to reasonable values: f_{cpu} varies between 99 and 471MHz, f_{bus} varies between 50 and 236MHz and f_{mem} varies between 99 and 133MHz. A total of 22 unique setpoints were used.

The electrical specifications for the PXA255 give the appropriate CPU core voltage for a number of frequency settings. Since this information does not specify the core voltage required for every possible frequency, a coarsely linear relationship between frequency and voltage was found and used to calculate the voltage for the unspecified frequencies. Furthermore, the S1F81100 power supply chip only allows discrete steps of 0.1V. For each frequency we picked the available voltage closest to a linear interpolation between frequency-voltage pairs specified in the processor's documentation (and confirmed that the processor operated correctly at that voltage). This does not introduce errors in our model, as it makes no assumption on the relationship between voltage and frequency.

The PXA255 provides two performance counters and one cycle counter. The configurable counters can each be configured to count one of 14 events, described in the PXA255 developer's manual [10].

All experiments were conducted in Linux 2.4.19 with patches to support the PXA255. Kernel modules and modifications were made to support the performance-monitoring unit, voltage scaling via the power supply chip, and frequency scaling based on `cpufreq`. The PMCs are read and accumulated after each scheduler invocation. The present implementation remains un-optimised for performance.

6.2 Energy Measurements

A device based on an MCP3909 energy-measurement IC [12] was built and tested for measuring the overall system power and energy. The MCP3909 measures both current and voltage using matched, simultaneously sampling, 16-bit sigma-delta ADCs. The current is measured via a $20m\Omega$ shunt resistor, and the MCP3909's built-in differential amplifier. The device has an accuracy of better than $1mA$ and $1mV$ for current and voltage respectively, giving a power accuracy of about $5mW$ or 0.4%.

A microcontroller monitors a trigger signal and accumulates continuous readings from the ADC when the trigger signal is active. This signal is controlled by a general-purpose IO line on the PXA255, which is driven high at the start of a benchmark, and low at the end. When the trigger signal falls, the energy measurement is complete and the microcontroller sends the accumulated values via USB to a host PC. The readings are later matched with data measured on the target system itself.

The power is measured between a laboratory power supply and the input to PLEB 2's power supply chip. This avoids PLEB 2's front-end power-supply circuitry (e.g. reverse-polarity protection, fuse, front-end buck converter, battery charger, etc). This is the same way a battery would power the device. We could derive the power-supply input power from the predicted battery power if needed.

6.3 Benchmarks

Both the time and energy models call for identical workloads to be executed at different frequencies. We achieve this via the use of end-to-end measurements of a number of benchmark suites. The workloads used are similar to those used in our previous work [22]. The validation of the model uses a benchmark set different from the calibration suite, in order to obtain a sound assessment of the model's predictive capability.

The essential requirement of each workload is that the same work must be done independent of the frequency setting. This means that the same instructions must be executed on the same input data.

Our **calibration suite** consisted of 37 benchmarks sourced from the MiBench suite, SPEC CINT95 and elsewhere. MiBench [9] is designed to be representative of many different classes of embedded systems. Several benchmarks (`adpcm`, `djpeg`, `stringsearch`, `susan_corners`, `susan_edges`, `tiff2bw`) were removed as their total execution time was too short to be useful (less than 0.25s at maximum frequencies). Two short-running ones (`adpcm` and

stringsearch) were modified to iterate a number of times in order to extend the overall run time. Two others (sphinx, pgp) were removed as their execution is (intentionally) non-deterministic, making them unsuitable for our purposes. tiff2rgba and tiffmedian were removed due to their unusually high system-call rate – managing the power consumed by the operating system is beyond the scope of this paper.

From SPEC CINT95, vortex was excluded because of memory constraints, go and m88ksim due to runtime errors, leaving 5 benchmarks. We used the "test" dataset and reduced the input data size for compress to 420000 bytes to reduce overall execution time. Several other benchmarks (celp, gzip, mpg123, vision, as described in our previous work [22]) were added for further workload variety.

The **validation suite** consisted of 10 benchmarks from the SPEC CPU2000-v1.3 CINT2000 suite. All benchmarks were included aside from mcf, for which there was insufficient memory, eon, for which the C++ libraries were required, and perlbmk due to runtime errors. Again, we used the "test" dataset to yield reasonable running times. gap was changed to use 32MB memory rather than 64MB. We also included some synthetic benchmarks in the validation, in order to test the accuracy of our model under selected extreme conditions. cpubound executes an unrolled loop of NOP instructions entirely in cache. membound and readbound execute an unrolled loop of out-of-cache writes and reads respectively.

All benchmarks were compiled or assembled using gcc 3.4.4 with softfloat and XScale tuning. The Linux kernel and ramdisk software were compiled using gcc 3.3.2.

7. RESULTS

We ran the benchmark suites described in Section 6.3 on the PLEB 2 platform and measured the energy consumed. Each benchmark was run at all 22 frequency setpoints. In order to sample a wider range of voltage settings, the calibration suite benchmarks were each also run at two constant voltages (1.3V and 1.5V) which are high enough for safe operation of the processor at all set points. The collected measurements form three data sets: calibration, validation and synthetics.

7.1 Time Model

The relative performance, as described by Equation 7, can be calculated given end-to-end measurements of a workload. We fit this equation to the measurements obtained from the calibration runs to obtain the coefficients α_i, β_i of Equation 4. In the calibration we always predict the performance at the maximum frequency $f' = f_{max}$.

We fitted the performance models to the calibration dataset using least-squares linear regression. Figure 3 shows the best-fitting models for up to 15 terms, based on the determination coefficient R^2 (although the parameter priority is the same with bic or other criteria). The figure shows R^2 values (indicated by colour and y-axis labels) starting with a single-parameter model in the lowest row, adding one parameter in each higher row. The parameters are listed as the x-axis labels, "Bus PMC$_x$" indicating PMC_x used for predicting C_{bus} etc. "Intercept" represents the x-axis intercept of the linear model.

The best two-parameter model uses DTLB misses (PMC4) for both bus and memory. Similarly, the best four-parameter model uses TLB misses and data cache misses (PMC11). The best six-parameter model is less clear-cut, either using instruction cache misses (PMC0), data-dependency stalls (PMC2), ITLB misses (PMC3) or data-cache-buffer-full stalls (PMC9). The good news from this graph is that in most cases, the same counters are selected

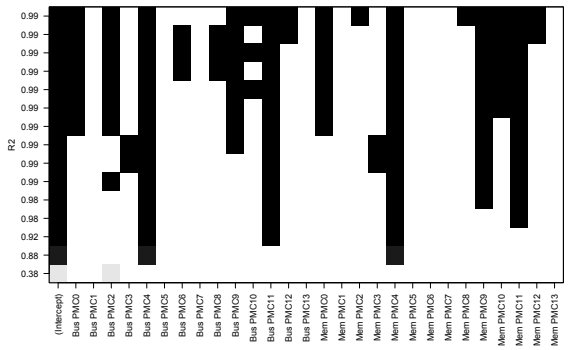

Figure 3: Parameter selection for the execution time model

for predicting C_{mem} and C_{bus}. This means that with n counters we obtain a good $2n$-parameter model.

We chose a 3-counter (6-parameter) model based on PMC4, PMC9, PMC11, because of its later utility in the power model. Adding more parameters does not appear to improve the quality of the model. Both the two- and three-counter models had an excellent determination $R^2 > 0.9999$, with all parameters significant. Predicting the performance of the validation benchmarks using these models gives a 6.1% maximum and 1.4% average error for the 2-counter model, and a 4.9% maximum and 1.6% average for the 3-counter model. The errors for the latter are plotted in Figure 4.

Checking the model for the extreme cases (synthetic benchmarks), the maximum error in the relative performance (i.e. the fraction of the speed at which the workload was run) for cpubound was 0.2% for the 3-counter model. In contrast, the memory-related synthetic benchmarks performed badly: membound yielded 155% error for the 3-counter model, while readbound yielded 11%. We consider the errors for the latter two to be indicative of the error for a pathological case (as opposed to the likely error for a real-world workload, which is much smaller). In particular, it is clear that the ability of the model to account for the pathological write-bound case (membound) is lacking, a reflection that the platform has simply no performance counters which accurately measure memory

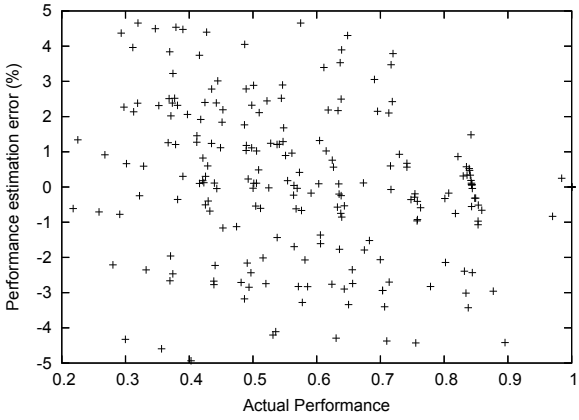

Figure 4: Estimation error for the 3-counter execution-time model

accesses, this has to be approximated by other counters. It would be possible to construct other pathological cases. For example, the execution time for a workload which was bounded by the instruction cache would be unlikely to be accurately estimated since the models outlined here do not examine instruction cache events. The set of workloads used for parameter selection and regression should be representative of the workloads to be predicted.

7.2 Energy model

7.2.1 Parameter Selection

Similar to the execution-time model, we performed a parameter selection for the power model presented in Equation 11. In order to maximise the number of data points available, the samples from each benchmark were used to predict the power for every other sample of that benchmark. The resulting calibration data set contained 142,296 entries. Compared to Equation 11, we also introduced a few extra parameters. Specifically we included, besides the $V^2 f_x$ terms, also al $V f_x$ and plain f_x terms, even though their physical interpretation is not obvious.

The results for models up to 20 parameters are presented in Figure 5. Again, we compare the models based on R^2.

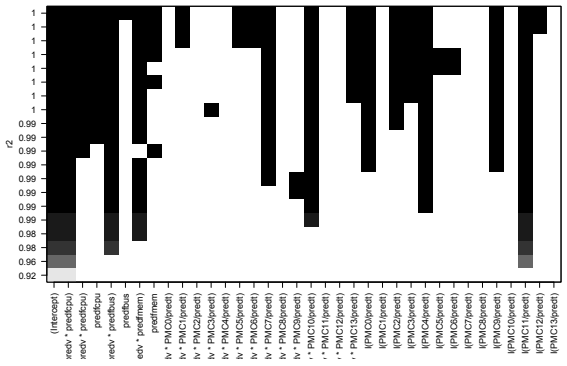

Figure 5: Parameter selection for the power model

The figure shows that the $f_{cpu}V^2$, $f_{bus}V^2$ an $f_{mem}V^2$ terms are all relevant. We see $f_{cpu}V$ and f_{cpu} also have an effect (most likely due to a voltage or frequency dependence in the leakage power). Of less importance is f_{mem}, despite it driving the external bus, we assume some co-linearity with $f_{mem}V^2$. This is un-surprising, given PLEB 2's memory frequency was only varied to three settings. The intercept is also significant, and represents the system's static power (the component of power which we assume to be constant).

From the terms which are proportional to V^2, and therefore related to the CPU core power, the parameter selections indicated instructions and cache accesses often had an effect on power. The selection in Figure 5 shows the importance of data cache accesses (PMC10). Cache accesses cause an increase in the CPU power because accessing the cache requires a large amount of circuitry to be active. Also, since each instruction (PMC7) takes an amount of energy to execute, the rate at which they are executed is positively correlated with power.

The external power is dominated by accesses to SDRAM, so the parameters unrelated to V^2 which are selected tend to attempt to approximate the number of accesses to main memory. Data-cache misses (PMC11) and Data TLB misses (PMC4) both have a strong effect on the model. The two parameters with the next strongest effect are data-cache-buffer related stalls (PMC9) and instruction cache misses (PMC0). PMC0, PMC4 and PMC11 are easily explainable as being indicators of memory accesses. PMC0 and PMC11 are both cache miss events. PMC4 is a Data TLB miss, which requires an access to memory (bypassing the cache) in order to walk the page table.

The behaviour of PMC9 in the parameter selection is less obvious. In the 6- and 7-parameter models it appears as with a V^2 term, but at higher models on its own. As noted elsewhere [4, 21], the counters available are not able to accurately estimate the number of actual accesses to main memory. This is because of `fill` and `pend` buffers between the cache and memory controller. These allow the processor to continue execution while the memory is accessed. If a second miss occurs, it will be counted, but the request will be serviced when the original access returns. Changing the frequency changes the likelihood of this event, and therefore the number of cache misses observed. We surmise that for the calibration data set, PMC9 helps to provides an estimate of this occurrence.

Section 7.2.2 examines the effect of using more or fewer counters. Unless otherwise stated, we use a four-counter model using data-cache accesses, data-TLB misses, data-buffer-full stalls and data-cache misses. Three of these counters are used in our three-counter time model. We used PMC9 without the V^2 factor, in accordance with the best selection for 8+-parameter models. We used $fcpuV^2$, $fbusV^2$ and $fmemV^2$, although we suspect that, given more frequency setpoints, a better relationship could be found.

7.2.2 Calibration and Validation

Models using between 3 and 6 counters were selected based on the parameter selection, again fitting to the calibration data set using least-squares linear regression. The model was then used to estimate the power at the sampled setpoint for the validation data. This estimate was compared with the measured average power for each benchmark. The rate terms in Equation 11 were calculated using the measured time. The results are presented in Table 1.

Counters	Param.	R^2	Max Err (%)	Avg Err (%)
1	4	0.9836	7.46	2.14
2	5	0.9871	6.94	2.31
3	6	0.9904	4.85	1.26
4	7	0.9922	3.78	1.16
5	8	0.993	3.68	0.92
6	9	0.9938	2.94	0.89
6	11	0.9947	2.75	0.77

Table 1: Regression and validation data for various models

While the PXA255 in PLEB 2 has only 2 performance counters, other variants, such as the PXA270, have four (usable for the same events). Furthermore, if on-line estimation for PLEB 2 required more than two measured events, the counters could be multiplexed (i.e. the events being counted would be periodically switched. Assuming no pathological cases, this would allow for an accurate estimation of all events considered, at the cost of the responsiveness of the analysis being reduced. Therefore, for the remainder of this paper we will use a four counter power model (which uses several of the same events as the execution time model of Section 7.1).

The error distribution for this model is shown in Figure 6. The system's static power was estimated from the model intercept and found to be 1.14W. The measured and estimated dynamic power were calculated by subtracting this static power component. The maximum error in the dynamic power was 20% and the average error was 6% for the validation dataset and 4-counter model.

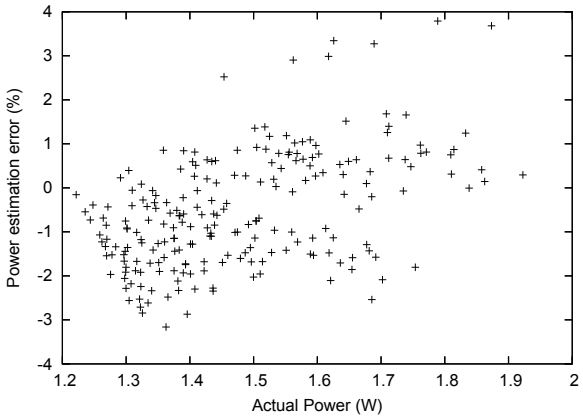

Figure 6: Estimation error for the 4 counter power model

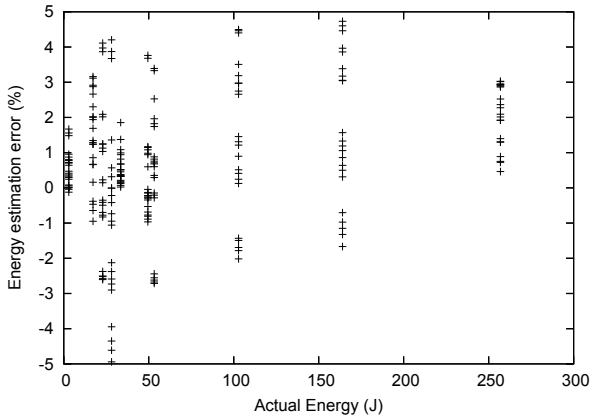

Figure 7: Maximum-frequency energy prediction error for the 4-counter energy model

The power was estimated for the three synthetic benchmarks using the same four-counter model. The estimates for the `cpubound` total power had a maximum error of 6%, while `membound` 56% and `readbound` 10% error. Again, we consider these benchmarks to indicate the worst cases for the power estimation model, since they are very unlike the benchmarks used for parameter selection and calibration and exercise features of the CPU which are unmeasureable by the performance counters. We consider the synthetic benchmarks to indicate the extreme cases for power estimation.

7.2.3 Prediction

The 3-counter execution-time model and 4-counter power model were combined to obtain 4-counter power and energy models. The power measured at the maximum frequency was observed to vary between 1.70W and 2.19W for all benchmarks. The 4-counter power model was used to estimate the power at the maximum frequency for the validation data set, based on each benchmark run (i.e. from every other frequency). Using the measured time as the basis for calculating the event rates (therefore isolating the error of the power model), the maximum error observed was 3.7% and the average error was 0.72%. Using the time estimated by the time model, a maximum error of 4.3% and an average error of 0.7% were observed.

The energy model was used to predict the energy at the highest CPU frequency setpoint, and the prediction was compared with the actual energy required to run the workload at that setpoint. Here, the energy prediction is made using the estimated time model. The maximum error was 4.9% and the average error was 1.5%. A plot of the error in energy prediction for the maximum frequency from all other frequencies is shown in Figure 7. Similar predictions could be made for any other setpoint.

8. CONCLUSIONS AND FUTURE WORK

This paper has presented a general performance-counter based model for the prediction of power and energy under voltage and frequency scaling.

We observed that a system's energy response to DVFS is highly dependent on the workload and have developed a technique that allows us to incorporate the workload-dependence through performance counters. We have furthermore demonstrated how the platform-specific parameters of the model can be obtained through a process based on linear regression between performance-counter readings and power predictions of a family of models.

We have performed a sound evaluation of the models by using a validation benchmark suite that is different from the calibration suite. It was found that on our target platform, a PXA255-based system, the energy use of a program at a certain frequency setpoint can, with a fair degree of accuracy, be predicted from observing the system at a different setpoint (although we found prediction errors could be very high in pathological cases). This gives us a tool that can be used in the future to let an operating system (OS) perform informed energy-management decisions.

However, it became clear that the two concurrent performance counters supported by the PXA255 are insufficient for this approach without multiplexing the counters over several events. Fortunately, more recent processors provide a larger number of performance counters.

While the accuracy of the predictions generally seemed acceptable, it is also clear that the set of events that can be monitored on the PXA255 is not sufficient go give a really accurate model of power consumption, as it was only possible to roughly estimate the number of bus transactions, or CPU stall cycles. We suspect that more suitable events could be provided by hardware manufacturers [16]. Since the limited number of performance counters do not capture the entire behaviour of the system, the choice of benchmarks used for parameter selection and model calibration is important in order to achieve good coverage of anticipated application behaviour.

The results presented in this paper enable a large amount of follow-on work. For one, we plan to show that our approach works reliably across a range of hardware platforms, including different CPU architectures and different performance-monitoring capabilities, but also different balances of static and dynamic power.

Then we plan to use our models as the underlying mechanism to enable proper OS-level system power management, especially when dealing with real-time constraints. This includes taking into account the selection of power states and the cost of using them, and designing suitable policies. This will also require extending the approach to deal with multi-tasking workloads, including context-switching costs and accounting for OS services. Adding extra execution contexts will not only increase cache miss rates, but also introduce the overheads of frequency switching (if two threads run at dissimilar setpoints).

Finally we would like to incorporate I/O into our power/energy model.

9. REFERENCES

[1] A. Acquaviva, L. Benini, and B. Ricco. Software-controlled processor speed setting for low-power streaming multimedia. *IEEE Transactions on CAD ICAS*, 20(11):1283–1292, Nov. 2001.

[2] F. Bellosa. The benefits of event-driven energy accounting in power-sensitive systems. In *Proceedings of the 9th SIGOPS European Workshop*, Kolding, Denmark, Sept. 17–20 2000.

[3] L. Bircher, M. Valluri, J. Law, and L. John. Runtime identification of microprocessor energy saving opportunities. In *Proceedings of the 2005 International Symposium on Low Power Electronics and Design*, 2005.

[4] G. Contreras and M. Martonosi. Power prediction for Intel XScale processors using performance monitoring unit events. In *Proceedings of the 2005 International Symposium on Low Power Electronics and Design*, San Diego, CA, USA, Aug. 2005.

[5] E. Duesterwald, C. Cascaval, and S. Dwarkadas. Characterizing and predicting program behavior and its variability. In *Proceedings of the 12th International Conferenece on Parallel Architectures and Compilation Techniques*, Sept. 2003.

[6] J. Flinn and M. Satyanarayanan. Powerscope: A tool for profiling the energy usage of mobile applications. In *Proceedings of the Second IEEE, Workshop on Mobile Computing Systems and Applications*, 1999.

[7] F. Fruth. Run-time energy characterization of the Intel PXA. Study thesis, Operating System Group, University of Erlangen, Germany, Apr. 2005.

[8] D. Grunwald, P. Levis, K. I. Farkas, C. B. Morrey III, and M. Neufeld. Policies for dynamic clock scheduling. In *Proceedings of the 4th USENIX Symposium on Operating Systems Design and Implementation*, pages 73–86, San Diego, CA, USA, Oct. 2000.

[9] M. R. Guthaus, J. S. Reingenberg, D. Ernst, T. M. Austing, T. Mudge, and R. B. Brown. Mibench: A free, commercially representative embedded benchmark suite. In *Proceedings of the 4th IEEE Annual Workshop on Workload Characterization*, Dec. 2001.

[10] Intel Corporation. Intel PXA250 and PXA210 applications processors developers manual. http://www.intel.com/design/pca/products/pxa255/techdocs.htm, 2005.

[11] T. L. Martin and D. P. Siewiorek. Nonideal battery and main memory effects on cpu speed-setting for low power. *IEEE Transactions on Very Large Scale Integration Systems*, 9(1):29–34, Feb. 2001.

[12] Microchip Technology Incorporated. MCP3909: Energy metering IC with SPI interface and active power pulse output. http://www.microchip.com/MCP3909, 2006.

[13] A. Miyoshi, C. Lefurgy, E. V. 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, New York, NY, USA, 2002. ACM Press.

[14] R. Nathuji, B. Seshasayee, and K. Schwan. Combining compiler and operating system support for energy efficient i/o on embedded platforms. In *Proceedings of the ACMWorkshop on Software and Compilers for Embedded Systems*, Dallas, TX, USA, 2005. ACM Press.

[15] R. Neugebauer and D. McAuley. Energy is just another resource: Energy accounting and energy pricing in the nemesis os. In *Proceedings of the 8th Workshop on Hot Topics in Operating Systems*, Schloss Elmau, Germany, 2001.

[16] J. Peddersen and S. Parameswaran. CLIPPER: Counter-based low impact processor power estimation at run time. In *Proceedings of the 12th Asia and South Pacific Design Automation Conference*, Yokohama, Japan, Jan. 2007.

[17] Seiko Epson Corporation. S1F81100 technical manual. http://www.epsondevice.com, 2003.

[18] K. Seth, A. Anantaraman, F. Mueller, and E. Rotenberg. FAST: Frequency-aware static timing analysis. *ACM Transactions on Embedded Computing Systems*, 5(1):200–224, 2006.

[19] D. C. Snowdon. PLEB 2 web site. http://www.ertos.nicta.com.au/hardware/pleb, 2006.

[20] D. C. Snowdon, S. M. Petters, and G. Heiser. Power measurement as the basis for power management. In *Proceedings of the 2005 Workshop on Operating System Platforms for Embedded Real-Time Applications*, Palma, Mallorca, Spain, July 2005.

[21] D. C. Snowdon, S. Ruocco, and G. Heiser. Power Management and Dynamic Voltage Scaling: Myths and Facts. In *Proceedings of the 2005 Workshop on Power Aware Real-time Computing*, New Jersey, USA, Sept. 2005. Unpublished proceedings, available from http://ertos.nicta.com.au/publications.

[22] D. C. Snowdon, G. van der Linden, S. M. Petters, and G. Heiser. Accurate run-time prediction of performance degradation under frequency scaling. In *Proceedings of the 2007 Workshop on Operating System Platforms for Embedded Real-Time Applications*, Pisa, Italy, July 2007.

[23] M. Waitz. Accounting amd control of power consumption in energy-aware operating systems. Diploma thesis, Operating System Group, University of Erlangen, Germany, Jan. 2003.

[24] M. Weiser, B. Welch, A. J. Demers, and S. Shenker. Scheduling for reduced CPU energy. In *Proceedings of the 1st USENIX Symposium on Operating Systems Design and Implementation*, pages 13–23, 1994.

[25] 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*, Grenoble, France, Oct. 8–11 2002.

[26] A. Weißel, B. Beutel, and F. Bellosa. Cooperative IO—a novel IO semantics for energy-aware applications. In *Proceedings of the 5th USENIX Symposium on Operating Systems Design and Implementation*, Boston, MA, USA, Dec. 2002.

[27] H. Zeng, C. S. Ellis, A. R. Lebeck, and A. Vahdat. Currentcy: Unifying policies for resource management. In *Proceedings of the 2003 Annual USENIX Technical Conference*, San Antonio, Texas, June 2003.