Power Measurement as the Basis for Power Management

David C. Snowdon, Stefan M. Petters and Gernot Heiser

National ICT Australia
and
School of Computer Science and Engineering
University of NSW, Sydney 2052, Australia
{daves,smp,gernot}@cse.unsw.edu.au

Abstract

Energy has become a critical component of computer system design, particularly in the embedded space where the source of energy is often finite. While hardware design has the more significant effect on the system's power consumption, well designed system and application software make an important contribution to controlling the energy consumed.

In order to optimise software systems to reduce energy consumption, feedback is required. Traditional techniques rely heavily on models of the system which have various disadvantages. We examine the benefits of using live power measurements using statistical sampling for both off-line and on-line feedback on application power consumption. A hardware platform is manufactured, operating system modifications made, and extensive validation carried out. We conclude that the idea shows promise and justifies further investigation.

1 Introduction

Computer power usage has become an important area of research for a number of reasons. High-performance systems are limited by problems with thermal dissipation, and portable and embedded systems are often supplied power from a limited source (batteries, solar panels). In both cases, energy efficiency is a key quality determining computer system utility. Energy awareness and management is critical in improving the energy efficiency of a computer system. This is loosely termed *power management*.

There are three key problems which power management research has attempted to solve. Generally, these problems are:

- determining how, and why, power is used in a computer system;
- configuring hardware to match power and performance requirements (dynamic voltage scaling and low power idle states have become standard on modern hardware);
- adapting software to use available resources efficiently.

We observe that it is useful to know, to some level of detail, how and why a computer uses power. Such information is used to evaluate and analyse the operation of power management algorithms, can help to optimise application and operating system code with respect to power consumption, and supports the generation of off-line schedules. We further hypothesise that, should the information be available at run-time, it would be useful as the basis for some classes of power management algorithms, particularly for intelligent throttling of system components (CPU via frequency scaling, hard disk via spin-down, etc).

This paper presents the details of an investigation into obtaining power usage information through direct measurement of the current supplied to the computer's processor core, memory and IO subsystems. Hardware and software infrastructure for making these measurements is developed, the overheads examined, and the accuracy of the system assessed.

The initial version of the system was used as an off-line analysis tool. The same methodology can be used to perform on-line measurements, giving feedback to the operating system and user-level processes, and allowing the system to block processes which exceed an energy quota. The effectiveness of these techniques is examined, leading to ways in which the information can be used, both for off-line analysis and for on-line accounting/power-management.

2 Previous work

A variety of energy-estimation techniques have been developed with a view to programmer feedback, power management research evaluation, and on-line accounting. Many of these techniques rely on indirect measurements coupled with a model of the system in order to estimate the power used. There are several disadvantages to a model-based approach with regard to power estimation:

- A sufficiently detailed model is required in order to obtain a given accuracy. Computational complexity must be traded with the detail and accuracy of the model, and sufficient information must be available in order to construct it. Circuit-level simulations require the actual circuit design, etc.
- Modifications and additions to the system require changes to the model — a significant engineering effort.
- The model must be verified against real-world measurements.
- Most established simulators concentrate on the CPU rather than the entire system.
- Models will inevitably miss details: the model of a hard disk is unlikely to take into account the physical condition and situation of the disk, which might affect power consumption.
- Similar to execution times the manufacturers are reluctant to provide the details required, since it may give advantageous information to competitors.

One advantage of model-based estimation techniques is that the parameters fed to the model are often useful in their own right.

Simulators are often proposed as an off-line analysis tool [5, 7, 10–12]. Typically, trace output from an architecture-level simulator (such as SimpleScalar [**?**]) is obtained, and an energy associated with each instruction in the trace. The energy used is usually pre-calibrated via measurement of the actual hardware, since it is rare that the detailed design information necessary to accurately determine this via circuit-level simulation is available.

Event-counter based techniques, typified by Bellosa and Weissel's work [14], use live data generated by CPU performance counters as the input to a model. The counters are configured to measure events which are significant to the energy consumption (cache misses, instructions retired, etc), and a model interprets these results to estimate the total CPU power consumption. The accuracy of the system is therefore determined by the amount of information available (the number of event counters and measured properties). The model used is typically simple and the simplifications can lead to inaccuracies in the estimation. These systems have the advantage that they can efficiently be used on-line, allowing the information obtained to be used by power management algorithms. This technique has only applied to CPUs, since performance counters are generally not available in peripherals. The only exception to this are memories, which could be observed, albeit indirectly, by counting cache misses and write back operations.

State-based accounting techniques such as those employed in ECOSystem [16] instrument operating system software to track the state of the CPU and its peripherals (e.g. for a disk, whether it is spun up or down and whether it is active or idle). The power in each of these states is pre-calibrated, and the time spent in each used to determine the energy consumed. In ECOSystem the energy is then accounted to the processes causing the device to transition out of its lowest-power state. These techniques have the advantage of being an all-software solution which can simulate the entire system, however they fail to capture any variation of the power within a given state. The accuracy of the technique therefore depends on the number of states (detail of the model).

Jejurikar and Gupta introduce a system which uses off-line and on-line analysis to reduce the energy usage of their applications [8]. The off-line part produces the slowdown factor in such a way that under the assumption that every task runs for its worst-case execution time (WCET) all deadlines are just met. In the real system deployment this will hardly be the case as most applications almost never run for their WCET. The on-line part takes advantage of this "gain time" to increase the slowdown in order to keep CPU utilisation high. Such an approach would complement the proposed measurement technique we are presenting here, which may be used to produce the input values for their optimisation. Similarly, AlEnawy and Aydin look at on-line and off-line methods [2]. Their results are produced by simulations rather than using direct measurements our system would enable.

An alternative to using model-based power estimation techniques is proposed by Flinn [6]. He uses statistical sampling techniques (as are widely used in sampling profilers such as Shark [3]). Measurements of the power consumed by a computer are taken periodically, along with the program counter, and process ID. This information is stored and later analysed by attributing each power sample to a process and to a symbol within the process' code. Although the sampling rate is slow in comparison to the CPU's clock rate, over time enough samples are attributed

to each piece of code (process/symbol) that a statistically significant average is obtained. This information is more detailed than either state-based or event-counter-based techniques can provide, and comparable with (and potentially more accurate than) results obtained via simulation. Flinn called his energy-profiler Powerscope.

Using these direct measurements counters the disadvantages of model-based approaches, however there are several problems which were not addressed in the original and subsequent Powerscope work.

One of particular importance is the inability of the system to account for background activity (activity which is not associated with the process which is running on the CPU) such as blocking disk reads and writes). In the original Powerscope system this activity is accounted to the running process rather than the blocked process which is actually responsible for the activity. It is impossible to distinguish between the power consumed by the disk (which should be attributed to the blocked process) and the power consumed by the processor (which should be attributed to the running process).

A related issue is not being able to understand how and where energy is being used in the system. For example, it is not possible to discern between energy consumed by the CPU and memory subsystems (without estimating via a system model).

Other problems include the cumbersome hardware setups (the original Powerscope work involved a second computer connected to an external multimeter in order to perform the power measurements. The low sampling rate which was used means very short functions are not measured accurately (owing to an effective low-pass filter at the measurement input). Lastly, the information can not be used at run-time, since the measurements are taken using a different computer.

3 Measurement system

Many of the problems identified in Powerscope can be avoided by building a computer with hardware support for taking the measurements. one of these problems is, for example, the attribution of background activities, such as network traffic or hard-disk data transfer, to the wrong process. By splitting up the measurements into separate entities for these devices and associating those with modules instead of processes, a deferred attribution to processes is possible.

For example, a system with a CPU, memory, network card, and hard disk, would measure the power consumed by each of these independently. The system can then associate the CPU measurement with the running process, the network card with processes which have submitted or received packets, the disk with processes using file systems, and the memory system with processes causing memory bus accesses (which may be associated with devices, owing to DMA).

3.1 The PLEB 2 Platform

PLEB 2 is a single-board computer based on the Intel XScale PXA255 [1]. It was custom-designed primarily as a reference to be used in embedded systems research, but secondarily as a platform for applications implementation.

The PXA255 was chosen as being representative of high-performance CPUs designed for 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 main processing core consists of the CPU, SRAM and flash memory. Three switching power supplies generate core, memory and IO power from lithium-ion battery voltage. A minimal set of peripherals (infra-red, USB, and serial port) are provided on-board, and supplied from IO power. An 8-bit microcontroller (an Atmel AVR) resides on-board in a supervisory role. This models a typical embedded system. Un-used pins on the XScale are connected to two connectors which allow for other peripheral electronics to be added.

Linux 2.4.19, Linux 2.6.8 and L4ka::Pistachio [9] have been modified to run on PLEB 2 hardware.

Of particular interest in this context are the device's features designed to support power management. The CPU core, and memory clock frequencies can be changed in (very roughly) 30MHz and 10MHz intervals respectively (the intervals are smaller for the lower frequencies), although not all combinations of clock frequency, bus frequency, memory frequency, etc. are possible.

One problem encountered with frequency scaling (changing the frequency on-the-fly to adapt to performance requirements, thereby saving energy) is that there is an overhead associated with changing frequencies. The XScale attempts to solve this by offering a *turbo mode* which is a second frequency mode. Changing between the run (normal) mode, and turbo mode is much faster than changing between arbitrary frequencies because the system can perform a synchronous switch between the modes, without having to disturb the memory controller, LCD controller, and other peripherals.

As well as being capable of setting its core clock frequency, the CPU can enter a number of low-power states. These states disable circuitry within the CPU: the more circuitry disabled, the longer it takes to re-activate. Therefore it is necessary to ensure that the energy saved by being in

the sleep for a period of time is enough to offset the energy used to sleep and wake up.

Techniques for voltage scaling have also seen a lot of attention in the literature (most notably by Weiser [13], but also others too numerous to cite) - at a lowered frequency, the CPU's core voltage can be reduced, allowing quadratic energy savings (at half the frequency, the system will use a quarter of the power). The power supply used on PLEB 2 supports setting the voltage between 0.8 and 1.5V in 0.1V increments. The chip communicates with the PXA255 via I2C (a bidirectional serial bus). Similarly, the memory voltage can be set to either 3.3V or 2.5V, depending on speed and peripheral requirements.

Lastly, the Micron SDRAM used can place itself in power-down and self-refresh modes. These low-power states can save significant amounts of power. The Intel flash memory used for non-volatile storage has power-saving features, but they are not controllable, and therefore are of little interest in terms of power management.

3.2 Power measurement hardware

In support of embedded systems research, PLEB 2 was designed with power-measurement hardware on-board. Each of the three power supplies (nominally for the CPU core, memory and IO)[1] are instrumented with current sensors. Each power supply is well regulated to its designated voltage, therefore the voltage is assumed to be constant and the current is proportional to the power (P=IV).

The microcontroller on-board has an integrated analogue-to-digital converter and can read the sensors at up to 15kHz. Since it can only measure one of the sensors at a time, this equates to a maximum of 5kHz on the individual sensors when all are measured at equal rates.

Samples are transferred from the microcontroller to the PXA255 as they are taken (as described further in Section 4). Communication between the microcontroller and the PXA255 is via I2C. This is a significant limitation since I2C transfers data slowly (400kbps). Thus, in order to avoid excessive overhead, the transfer of each measurement requires several interrupts (one per byte — each measurement requires two data bytes to be transferred, along with the I2C bus' addressing byte). Furthermore, the maximum sampling rate is limited by the rate at which data can be transferred between the processors.

Figure 1 details the protocol for taking measurements: once enabled, the microcontroller interrupts the PXA255 periodically. The PXA255 initiates a measurement by asserting an interrupt line on the ATMEL chip. After a fixed delay the measurement of one of the three sensors is started. At the same moment the interrupt at the PXA255 is asserted. The interrupt handler records the state of the system (PC, PID, etc.) when the interrupt occurred. Once the analogue-to-digital converter has completed, the microcontroller interrupts the XScale a second time, triggering the XScale to start a transfer using I2C by sending a read command over the bus. This generates three further interrupts. After a short pause (which controls the sampling frequency) the microcontroller moves on to the next sensor to be sampled and the process repeats. The data transferred is stored by the XScale along with the PID and PC information previously recorded.

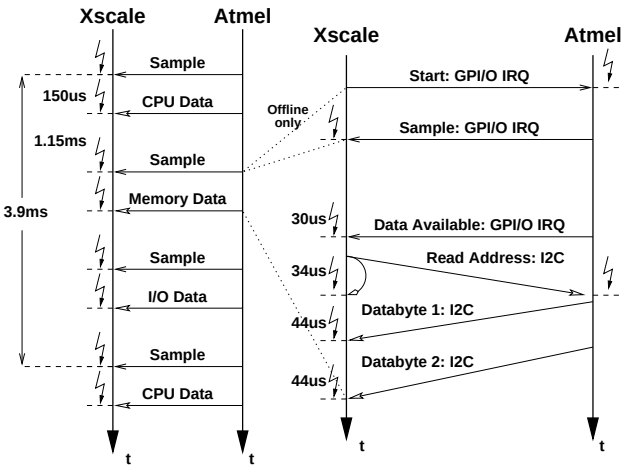

Figure 1. On-board communications timing of on-line analysis

4 Off-line analysis

The experiments presented in this paper were conducted using Linux 2.4.19 because of its immediate availability. Kernel modules, as well as minor changes to the kernel itself, were used to implement communications with the microcontroller, the off-line analysis and energy accounting and budgeting (Section **??**).

Off-line analysis in this context implies that the system collects data at run-time, and stores it for later examination. Power and time used by each process running on the system, as well as each function within the process and its shared libraries can be obtained from the data collected [6].

The off-line analysis facility is based on a port of the Powerscope code to the PLEB 2 platform which has been

[1] Note that, should peripherals (such as a network interface) be connected to the system, they will be connected to one of the three power supplies. This breaks the power-supply-per-device concept.

Benchmark	No profiling	Off-line	% overhead	On-line	% overhead
gzip	10.025	10.83	8.03	10.784	7.57
mpg123	30.256	31.213	3.16	31.071	2.69
vision	54.664	55.902	2.26	55.803	2.08
celp	85.397	87.34	2.28	87.17	2.08

Table 1. Time Overhead Introduced by the Measurement System in seconds

extended in order to take advantage of the extra hardware features available.

The Powerscope framework obtains samples via the protocol outlined in Section 3.2. It stores the samples for later analysis via a user-level daemon which reads from a kernel buffer (un-modified from the original implementation). The tool designed to analyse the data was modified to accommodate the three current sensors which do not sample concurrently.

This arrangement has a number of advantages over the original Powerscope implementation:

- Three current sensors are sampled, giving a user insight into how power is used in the system (it is possible to distinguish between memory and CPU power, for example). Further sensors can be added easily within the same software infrastructure.

- Because each of the three current sensors are connected to measure the major functional units, asynchronous activity (e.g. IO) can be accounted to the correct process and code. (ie. we can distinguish between background activity and activity directly correlated with the program counter and present process ID).

- The device is an integrated unit with no external apparatus required (Powerscope used a second computer and multimeter to reduce overhead on the system being profiled). This makes using the tool as easy as any other profiling tool. This also means there is only one data file which needs to be analysed, saving the need to move copy samples and data files to the same location.

The information gathered can be used in various ways. One way would be to guide the trade off between memory hierarchy and performance. Applications depending heavily on the CPU might benefit from the increased reuse of previous computation results stored in main memory, while applications with a large memory-bandwidth requirements can be optimised by recomputing values instead of relying on results stored in memory. In such a way the performance and energy usage could be optimised.

5 On-line analysis

5.1 Energy accounting

Because the XScale is set up to receive its own power measurement data, the information can be used on-line at run-time. The method of receiving data is very similar to the off-line system. For each sample, the value obtained is accumulated in Linux's process control (task) structure, and a field indicating the number of samples is incremented. Using this, the information is made available at user-level via the Linux `/proc` interface.

The method of taking direct measurements of the power consumed has numerous advantages over other methods of estimating the power consumed on-line:

- It does not employ a model, and therefore is not hindered by inaccuracies in that model. Furthermore, the extensive development time required to build an accurate model is avoided.

- Computation associated with accurate model-based simulations effectively prohibits their use for on-line power estimations.

- When comparing with state-based power estimators [15], which are often used for on-line power management in the literature, the measurement-based system can capture variation within a single state (for example, network interface power will vary greatly depending on whether it is sending, receiving, or both. The likelihood of these states can not be predicted by the operating system. Furthermore, the energy expended per packet will depend on the availability of the network).

- The approach does not only cover the CPU, but all the components within a system which are usually not covered in indirect measurement or simulator-based approaches.

- It is also possible, with little effort, to extend the approach to charge background IO activity to the process

	Powerscope CPU	LEA CPU	Powerscope Mem	LEA Mem
copymem	0.306	0.308	0.347	0.352
fillmem	0.310	0.314	0.405	0.412
fp-exercise	0.315	0.318	0.211	0.212
add_bench	0.268		0.211	

Table 2. Comparison of some typical results for on-line (LEA) and off-line (Powerscope) measurements. All measurements in Watts.

initiating this activity, rather than to whichever process is running during this background activity. This gives similar capabilities (with better accuracy) to the state-based currentcy system [15].

5.2 Energy budgeting

The on-line accounting technique has been used to implement an energy budgeting system. The information available allows the operating system to make scheduling decisions based on energy related criteria. The implementation is similar to the currentcy approach (cf. [15]) described in Section 2.

The OS process control structure is augmented with an energy remaining and energy budget field. The energy accounting system is used to decrease the energy remaining. Periodically (in the present implementation, once per second), the energy remaining field is reset to the budget value. The scheduler was modified to ignore processes with a negative energy remaining field. This halts processes whose budget has been exhausted until it is replenished.

The system is able to control the processes' power using direct measurements, rather than state-based accounting, as was deployed in the currentcy work. This allows the processes to be throttled more accurately. A Linux `/proc` interface allows to set the budget for each process.

The energy budgeting system allows control over how much energy processes use. This is a mechanism by which power management algorithms can throttle processes (ignoring quality of service constraints). Desired goals achievable include obtaining a desired battery lifetime, or maintaining a maximum CPU temperature. Further work will revolve around validating the energy budgeting system and leveraging the infrastructure to implement power management algorithms.

Throttling the processes via this energy budgeting technique is a mechanism rather than a power management algorithm. In order to meet deadlines and other quality of service objectives, the processes would have to degrade gracefully. This degradation is likely to be application specific, although it could potentially be built into middleware.

Techniques such as voltage scaling and the use of low-power processor modes are complementary and can be used to eliminate idle time and increase the "work" done by the system per joule.

6 Results

Of major concern is the perceived overhead of taking measurements, both in terms of power and time. We have chosen four benchmark programs to discuss the impact of the measurement system on the application:

- *gzip* represents a compression algorithm, which may be used to reduce memory footprint of data — in this case, a 1.4MB MP3 — the output is discarded;
- *mpg123* is an MP3 player operating on the same 1.4MB MP3 file and is representative of a typical multimedia application — the output is discarded;
- *vision* is computer vision software, which uses a low resolution (128x128) greyscale image and identifies the type, location and orientation of an object within the image;
- *celp* is a codebook excited linear predictor and has been adapted from version 3.2 of the US DoD's Federal-Standard-1016 implementation for a lossy speech compression algorithm.

The benchmark applications cover a wide range of embedded applications. Compression algorithms are common to reduce the amount of data to be transmitted over low bandwidth interconnect and field buses. Multimedia applications like the MPEG decoding example mpg123 are common in most 3G phones and PDAs. Similarly the celp example stresses audio compression technique for mobile communication over a low bandwidth carrier. The vision example is typical for software used in industrial manufacturing automation involving simple, low resolution and robust image processing software.

The data presented in Table 1 shows the time overhead introduced by the measurement system with the main processor running at 199MHz, the system bus at 99MHz, and the memory bus at 99MHz. It suggests that the impact of the off-line measurements is not unreasonable. The overhead comes from three sources: the Linux interrupt handling code, the measurement system interrupt handling code, and the associated cache-related costs of running these two. The comparably large impact on `gzip` can be explained by its heavy memory and cache dependency. The Linux interrupt overhead code is much larger than the actual measurement system code. One possibility is that Linux pollutes the caches, badly affecting the working set of `gzip`. Another possibility is that the interrupt code and page mapping are evicted from the cache and TLB when running memory-intensive applications, leading to overhead when the interrupt is triggered.

In the case of the on-line system, the overheads presented in Table 1 are not unreasonable for a prototype system. `gzip` is adversely affected in the same way as in the off-line measurements. The on-line measurement system shows slightly lower overhead than that of the off-line system as a result of its not having to store large amounts of data (a running total, rather than a complete history, is maintained), and smaller interrupt handler code size.

Future versions of the system will make use of the XScale's *fast interrupt queue* (FIQ) vector, avoiding overheads associated with the Linux interrupt code, while at the same time reducing interrupt latency and improving the measurement accuracy because the sampled program counter will be better synchronised with the actual measurement. Furthermore, the interrupt handler could be pinned in the cache and TLB, avoiding the particular effect on cache-intensive applications.

The measurement based monitoring technique may be used for real-time systems, especially when the interrupt is moved to a separate interrupt queue, the TLB entry pinned and handler is locked into cache. In the context of real-time systems the off-line measurements may be obtained at the same time as the traces for a measurement based worst-cases execution-time analysis as in [4]. This would make effective use of the test scenarios created.

It is necessary to take into account the potential latency added by other interrupt service routines. We believe that it is safe to assume that enabling the measurement system does not extend the WCET of any task being analysed. The testing undertaken may well be used in a measurement based WCET approach, as described by Bernat et al [4].

The accuracy of the measurements has been validated by checking:

- **sanity**: for a variety of benchmarks and combinations of benchmarks, measuring the total average current consumed at the input and comparing with the total power given by the measurements;
- **proportionality**: running benchmarks with a consistent power consumption and comparing the ratio between the CPU and memory power for both the output of the measurement system and the voltage presented to the microcontroller by the current sensors;
- **consistency**: running different combinations of benchmarks and checking consistency of the measured results (i.e. in the absence of cache effects or other cross-coupling of the process' power, the measured power should be the same independent of what programs are running concurrently).

In order to measure the system's instantaneous power consumption, it was necessary to use artificial benchmarks which hold the power consumption at a constant level (while that benchmark is running).

- `fillmem` repeatedly fills a 1MB block of memory with meaningless numbers.
- `copymem` repeatedly copies a 1MB block of memory.
- `fp_exercise` performs some CPU intensive operations.
- `mul_bench` repeatedly executes the `mul` instruction.
- `add_bench` repeatedly executes the `add` instruction.

In order to obtain a larger variety of powers to be measured, the frequency of the main processor core was adjusted to 99MHz, 199MHz, and 398MHz. The processor's internal bus was also adjusted according to half the core frequency.

There is some advantage given to the measurement system by keeping the benchmarks at a constant power: bandwidth limitations and mis-alignment of the analogue and digital samples will have less impact. This could be further examined by comparing identical functions run in different programs, or comparing the total energy consumed rather than the power. However, in these cases, the measurement system accuracy will be reduced because of a smaller number of samples. Furthermore, as the size of the entity being measured is reduced, the assumption that the power of the entity is not affected by the surrounding program (or programs) becomes less valid. Lastly, since it is unlikely that the entity would be passed the same input each time, its power would further vary. The latter two are not inaccuracy

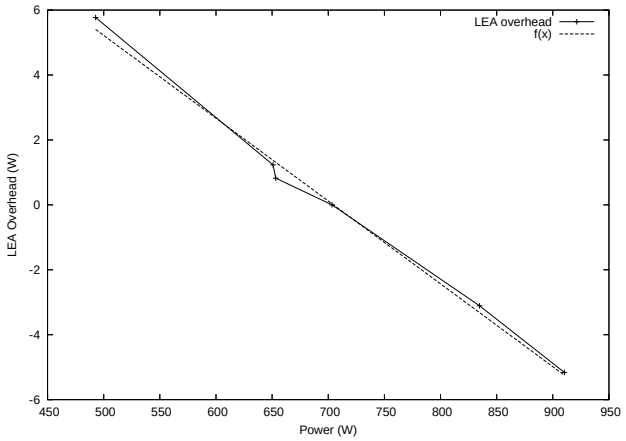

Figure 2. Overhead vs. actual power

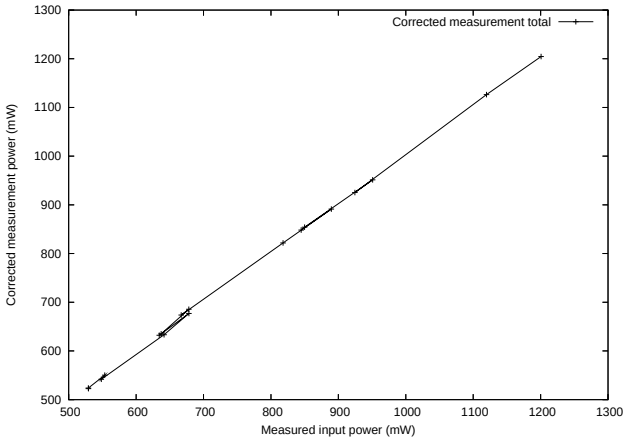

Figure 3. Measured vs. input power

in the measurement system, but a natural variation in the power. Further validation forms part of the future work in this area.

Figure 2 compares the power overhead with the actual system power without the measurement system running. The extra power used by the measurement system is found to be approximately constant, with a similarly constant percentage of the time executing. This is consistent with data, since there will be no change in input power if the measured software is using the same power as the measurement system. (i.e. the power will be constant). Mathematically:

$$P_{meas} = (1 - r)P_{in} + rP_{sys} \quad (1)$$

where P_{meas} is the power when the measurement system is running, P_{in} is the power when its not running, and r and P_{sys} are constants describing the proportion of time running the measurement system and the measurement system power respectively. Fitting this model to the data using least squares, we find that the measurement system uses approximately $C = 700mW$ for $r = 2.7\%$ of the time. This is approximately the same as thees time overheads shown in Table 1.

A sanity check was performed by comparing the input power (obtained by measuring the input current and voltage with two multimeters) with the sum of the (CPU, memory and IO) measurements for a given constant-power benchmark as given by the on-line measurement system. However due to the nature of the circuit the sum of the power consumed by the sum of the CPU, memory and IO power will not equate with the input power. There are several sources of power dissipation which must be considered in order to compare the two sets of measurements:

- The system uses several DC-DC converters to convert from the main supply voltage to the CPU, memory and IO voltages required. These converters have an efficiency which varies with the current and difference in voltage. For simplicity we assume a constant efficiency for each converter.
- A small amount of current consumed on the IO line. While PLEB2 has been designed to allow this line to be monitored, the components to do it were not available. The supply was physically measured using a current meter, and shown to be 20mA.
- There is a linear regulator to supply a clean voltage to the XScale's phase-locked loop, and another to supply the DC-DC converter logic.
- Other power drains are not accounted by the measurement system. Their power dissipation varies with the main supply voltage. We suspect these are resistive loads within the power-supply circuits.

For the purposes of checking the measurements, a simplified model was developed and fit to the measured values.

$$P_{in} = \frac{1}{\eta_{cpu}}P_{cpu} + \frac{1}{\eta_{mem}}P_{mem} + \frac{1}{\eta_{io}}P_{io} + \frac{V_{in}^2}{R_{DCDC}} \quad (2)$$

where η represents efficiency, P represents power, V represents voltage, and R_{DCDC} represents an equivalent resistive load in the DC-DC converter chip. DC-DC converter quiescent power, the power consumed by the linear regulators, and non-linearities in η are ignored for simplicity.

The data was fit to the above model numerically using least squares. The efficiency of the DC-DC converters was found to be approximately $\eta_{cpu} = 86\%$ and $\eta_{mem} = 95\%$.

The rated efficiency in the data sheet for the DC-DC converter circuit is 90% (the figures could be distorted by sources of power loss not considered in the model).

Figure 3 compares the corrected total power against the input power (measured with two multimeters) — i.e. each side of Equation 2. The maximum error between the two for these measured cases is 8mW (1.29%), showing that the measurement system is indeed making sane measurements (much of the error is likely to stem from the simplified model used to compare the input and measured power).

The defining feature of the measurement system is the ability to distinguish between the power used by different pieces of software running concurrently. This ability is proven by running several benchmark processes concurrently (i.e. with the Linux scheduler switching between them). Each benchmark should cause a near constant power draw during its period of execution. Varying the benchmarks which are run concurrently changes the system's average power consumption. The measurement system should be capable of distinguishing the different processes' constant power in each case.

Tables 3 and 4 show consistency between the measurements. Each of six tests was formed via a combination of one or more of the benchmarks and the power measured using the off-line system. It can be seen in the CPU results that the system varies by 2mW(0.6%) for the CPU power, and 9mW(2.6%) for the memory power (for this small sample size). The small variation in memory power can be explained by the increased cache misses caused by running concurrent processes. This effect varies due to cache placement, concurrent cache refills and write-back.

While these measurements presented were performed using the on-line system for convenience, the conclusions should apply to off-line measurements (i.e. the off-line and on-line results should be equal). Table 2 shows typical measurements for the constant-power benchmarks measured using both the off-line and on-line systems for comparison.

In summary, it was shown that the measurement system measurements can be equated with the observed input power. Then, that the measurements are consistent when run with a variety of benchmarks. Lastly we present some samples from both the on-line and off-line systems for comparison. We conclude that the system is useful as a tool for making power measurements.

7 Conclusions and future work

Direct power measurements, correlated with the in-system activity, provide a good way to obtain information to be used in analysing power use in computer systems. The implementation presented has low overheads and provides accurate results. It is easier and neater to use than previous implementations. The system can be used for off-line static analysis, and adds support for on-line accounting and budgeting.

A major advantage of the proposed approach over previous work is the ability to measure from more than one current sensor, allowing accounting for background activity, as well as more detailed information about how power is used in the system. Further advantages include not having a requirement for a detailed system model.

Future work will investigate several ideas:

PLEB 2 was designed as a general purpose research platform, and so was designed with the basic power monitoring features described in the previous sections. Given experience with PLEB 2 and a greater knowledge of the requirements placed on the measurement system, more appropriate hardware could be designed. An FPGA, rather than a microcontroller, could be used to coordinate current measurements. This would allow significantly reduced overheads, since the link via I2C could be replaced by a connection directly to the PXA255's memory bus, speeding and simplifying the data transfer. The FPGA could perform any necessary integrations or scaling.

Instead of measuring the current supplied by each power supply, a current sensor could be installed per device, allowing the system to measure the current consumed. This would mean each IO device would be individually monitored allowing users of the off-line analysis tools to understand how and why each device consumes power as well as allowing the on-line tool to accurately account for background activity.

Energy accounting could be done in the FPGA in order to improve the accuracy and and reduce the measurement system overheads. An FPGA with memory could be informed of a context switch, allowing it to track the power consumed by running processes without interrupting the system.

Applying the techniques discussed would both validate the ideas, and provide useful feedback about the behaviour of a typical system. Two possibilities are: compare a number of proposed dynamic voltage scaling techniques to determine how they perform at a system level (rather than using the CPU-specific power estimations), analysing the power consumption of a range of benchmarks, and integrating with timing analyses.

A more detailed investigation of the sensitivity to sampling frequency and process power variation would also be desirable.

Test	1	2	3	4	5	6
copymem	0.306		0.307	0.308		0.307
fillmem		0.310	0.311	0.311		0.311
fp_exercise				0.315	0.315	0.315
add_bench					0.268	

Table 3. Comparison of CPU power measurements by Powerscope for four benchmarks in six combinations. All measurements in Watts.

Test	1	2	3	4	5	6
copymem	0.347		0.349	0.340		0.340
fillmem		0.405	0.405	0.405		0.397
fp_exercise				0.211, 0.211	0.211, 0.211	0.211
add_bench					0.211, 0.211	

Table 4. Comparison of memory power measurements by Powerscope for four benchmarks in six combinations. All measurements in Watts.

References

[1] Intel PXA250 and PXA210 applications processors developer's manual. http://www.intel.com/design/pca/products/pxa255/techdocs.htm, 2005.

[2] T. A. AlEnawy and H. Aydin. On energy-constrained real-time scheduling. In *Proceedings of the 16th Euromicro Conference on Real-Time Systems*, Catania, Italy, July 2004.

[3] I. Apple Computer. Tools - performance and debugging. http://developer.apple.com/tools/performance/, 2005.

[4] G. Bernat, A. Colin, and S. M. Petters. WCET analysis of probabilistic hard real–time systems. In *Proceedings of the 23rd Real-Time Systems Symposium RTSS 2002*, pages 279–288, Austin, Texas, USA, Dec. 3–5 2002.

[5] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: a framework for architectural-level power analysis and optimizations. In *the proceedings of the 27th Annual International Symposium on Computer Architecture (ISCA)*, pages 83–94, 2000.

[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] S. Gurumurthi, A. Sivasubramaniam, M. J. Irwin, N. Vijaykrishnan, M. T. Kandemir, T. Li, and L. K. John. Using complete machine simulation for software power estimation: The softwatt approach. In *Proceedings for the 8th International Symposium on High Performance Computer Architecture*, pages 141–150, 2002.

[8] R. Jejurikar and R. Gupta. Optimized slowdown in real-time systems. In *Proceedings of the 16th Euromicro Conference on Real-Time Systems*, Catania, Italy, July 2004.

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

[10] T. Simunic, L. Benini, and G. D. Micheli. Cycle-accurate simulation of energy consumption in embedded systems. In *Design Automation Conference*, pages 867–872, 1999.

[11] A. Sinha and A. Chandrakasan. Jouletrack —a web based tool for software energy profiling. In *Design Automation Conference*, pages 220–225, 2001.

[12] V. Tiwari, S. Malik, and A. Wolfe. Power analysis of embedded software: A first step towards software power minimization. *IEEE Transactions on VLSI Systems*, 1994.

[13] 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.

[14] 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 CASES'02*, 2002.

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

[16] H. Zeng, X. Fan, C. Ellis, A. Lebeck, and A. Vahdat. ECOSystem: Managing energy as a first class operating system resource. In *Tenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS X)*, 2002.