March 2002

WRL Research Report 2002/1

Energy-driven Statistical Sampling: Detecting Software Hotspots

Fay Chang
Keith I. Farkas
Parthasarathy Ranganathan

COMPAQ **Western Research Laboratory** 250 University Avenue Palo Alto, California 94301 USA

The Western Research Laboratory (WRL), located in Palo Alto, California, is part of Compaq's Corporate Research group. WRL was founded by Digital Equipment Corporation in 1982. We focus on information technology that is relevant to the technical strategy of the Corporation, and that has the potential to open new business opportunities. Research at WRL includes Internet protocol design and implementation, tools to optimize compiled binary code files, hardware and software mechanisms to support scalable shared memory, graphics VLSI ICs, handheld computing, and more. As part of WRL tradition, we test our ideas by extensive software or hardware prototyping.

We publish the results of our work in a variety of journals, conferences, research reports, and technical notes. This document is a research report note. Research reports are normally accounts of completed research and may include material from earlier technical notes, conference papers, or magazine articles. We use technical notes for rapid distribution of technical material; usually this represents research in progress.

You can retrieve research reports and technical notes via the World Wide Web at:

http://www.research.compaq.com/wrl/

You can request printed copies of research reports and technical notes, when available, by mailing your order to us at:

Technical Report Distribution
Compaq Western Research Laboratory
250 University Avenue
Palo Alto, CA 94301 U.S.A.

You can also request reports and notes by sending e-mail to:

WRL-techreports@research.compaq.com

Energy-driven Statistical Sampling: Detecting Software Hotspots

Fay Chang[1], Keith I. Farkas[2], and Parthasarathy Ranganathan[2]

[1] Systems Research Center, Compaq Computer Corporation
130 Lytton Ave., Palo Alto, California 94301
[2] Western Research Lab, Compaq Computer Corporation
250 University Ave., Palo Alto, California 94301
{Fay.Chang, Keith.Farkas, Partha.Ranganathan}@Compaq.com

This paper was presented at the 2002 Power Aware Computing Systems Workshop (PACS) and is to appear in the formal workshop proceedings, which are to be published by Springer-Verlag Heidelberg as part of their Lecture Notes in Computer Science series.

Energy-driven Statistical Sampling: Detecting Software Hotspots

Fay Chang[1], Keith I. Farkas[2], and Parthasarathy Ranganathan[2]

[1] Systems Research Center, Compaq Computer Corporation
130 Lytton Ave., Palo Alto, California 94301
[2] Western Research Lab, Compaq Computer Corporation
250 University Ave., Palo Alto, California 94301
{Fay.Chang, Keith.Farkas, Partha.Ranganathan}@Compaq.com

Abstract. Energy is a critical resource in many computing systems, motivating the need for energy-efficient software design. This work proposes a new approach, *energy-driven statistical sampling*, to help software developers reason about the energy impact of software design decisions. We describe a prototype implementation of this approach built on the Itsy pocket computing platform. Our experimental results of 14 benchmark programs show that when multiple power states are exercised, energy-driven statistical sampling provides greater accuracy than existing time-driven statistical sampling approaches. Furthermore, if instruction-level energy attribution is desired, energy-driven statistical sampling may provide better resolution. On simple handheld systems, however, many applications may exercise only a single power state other than idle mode. In this case, time profiling may sufficiently approximate energy profiling for the purpose of assisting programmers, without requiring any hardware support.

1 Introduction

Energy is a critical resource for many computing systems, spurring the desire for energy-efficient software. Unfortunately, most current systems expose only coarse-grained information about energy consumption. Therefore, when programmers reason about the energy impact of their design decisions, they tend to rely on intuition. This intuition may be misleading if programmers have not internalized an accurate model of the energy cost of different operations or properly accounted for the relative frequency of the operations that take place.

Researchers have proposed a variety of tools that, by exposing more information about a system's energy consumption, could help programmers develop software that is more energy-efficient. Many of these tools estimate energy consumption by multiplying activation counts for various activities (e.g. executing a particular type of instruction) with estimates of the energy consumed to perform each activity [3, 4, 10]. One advantage of this approach is that it can leverage activation counts for background activities to more easily attribute energy consumed asynchronously. However, the approach has the disadvantage of requiring system-specific knowledge about what activities should be counted, and system-specific estimates of the energy consumed to perform each such activity.

In this paper, we describe and assess a new tool for helping programmers evaluate the energy impact of their design decisions. Our tool relies on statistical sampling, an alternative approach that does not require any system-specific information. Statistical sampling tools introduce a periodic source of interrupts. When each such interrupt is serviced by the system, the tool records a *sample*, which contains information about the system's state, such as the program counter of the interrupted instruction. The samples gathered during a given time period can then be analyzed to produce estimates of the system's behavior during that time period. In particular, our tool enables programmers to quickly determine the energy required to execute some software and the energy hot spots within that software. With this information, programmers may then employ techniques, such as those described by Simunic et al. [11], to improve energy efficiency.

Previous statistical sampling tools use sampling interrupts that are separated by a fixed amount of time (possibly with some small variation to diminish the risk of synchronization) [1, 6, 7]. In one approach used by these tools [1, 6], the distribution of the samples recorded by the tool constitute a *time profile* (an estimate of the proportion of time spent executing each instruction in the program). Previous work has demonstrated that time profiles can greatly assist programmers in reducing execution times [1]. However, since duration and energy consumption are not always proportional, time profiles can mislead programmers that are trying to reduce energy consumption.

An alternate time-driven sampling approach, PowerScope [7], adjusts for incongruities between duration and energy consumption by including in each sample an estimate of the system's instantaneous power consumption. This feature allows PowerScope to weight each sample by its instantaneous power measurement in order to produce an *energy profile* (an estimate of the proportion of energy spent executing each instruction).

In contrast, our tool uses sampling interrupts that are separated by a fixed amount of energy consumption (which we refer to as the *energy quanta*). Thus, the distribution of samples recorded by our energy-driven sampling tool directly constitutes an energy profile. Conceptually, with a sufficiently small energy quanta for our tool and a sufficiently high sampling rate for PowerScope, both of these tools should produce the same energy profile. However, because the sampling rate of our tool varies with the system's power consumption, our tool can deliver more accurate energy profiles for a given amount of overhead. Moreover, PowerScope's weighting scheme for generating energy profiles assumes that instantaneous power consumption approximates the average power consumption since the last sample. Measurements of our test system reveal large and rapid variations in power consumption, suggesting that this assumption may introduce significant inaccuracies. Finally, since PowerScope will continue to collect samples at the same rate even when the system is consuming relatively little energy, such as when the processor is in a light sleep mode, PowerScope's sample collection can significantly affect the actual energy profile of the system.

The key contributions of this paper are three fold.

- First, we introduce a new approach – energy-driven statistical sampling – for evaluating the energy impact of software design decisions.

- Second, we describe a prototype implementation of this approach for the Itsy pocket computing platform [9], and present experimental data that validates this implementation.
- Third, we present the results of an energy-consumption study of 14 benchmarks that compares energy-driven statistical sampling to existing time-driven statistical sampling approaches (both simple time profiling and the PowerScope approach). Our results show that, for a given sampling rate, energy-driven statistical sampling provides better resolution at an instruction level. It also provides greater accuracy when there are multiple power states, such as would exist in a system with dynamic voltage/frequency scaling. On the other hand, when there is only a single power state other than idle mode (which was the case for most of our benchmarks on our test system), time profiles may sufficiently approximate energy profiles for the purpose of assisting programmers.

The rest of this paper is organized as follows. In Section 2, we describe our prototype implementation of energy-driven statistical sampling and an error analysis of this prototype. Then, in Section 3, we describe our prototype implementation of the time-driven statistical sampling approaches, the methodology we used in our empirical comparison, and its results. In Section 4, we expand on the differences between sampling-based approaches and activation-model approaches. Finally, in Section 5, we conclude and discuss ongoing and future work.

2 Energy-driven Statistical Sampling Prototype

We have built a prototype implementation of energy-driven statistical sampling for the Itsy version 2.0 pocket computing platform [9]. In this section, we describe the hardware and software components of our prototype and the experimental results validating this prototype.

2.1 Hardware

As illustrated in Figure 1, we replaced the Itsy's battery with a power supply, and interposed an *energy counter* between the power supply and the Itsy electronics. We used a power supply rather than the battery to simplify our experimental procedure and to ensure that the Itsy's power efficiency stays constant during experiments; this issue is discussed further in Section 2.3. The purpose of the energy counter is to generate an interrupt whenever a predetermined amount of energy has been consumed. The energy counter operates as follows.

As current i_s is drawn from the power supply by the Itsy, a current mirror comprising a resistor and a current-sense amplifier generates a current $i_m = \alpha \times i_s$ (where $\alpha = 1.5 \times 10^{-3}$ in our implementation). This current i_m deposits charge on the positive plate of a capacitor, which acts as a current integrator. When the voltage across the capacitor plates reaches $\frac{2}{3}$ of the supply voltage of the monostable multivibrator (V_x), the multivibrator generates an output pulse P, and then discharges the capacitor to a voltage of $\frac{V_x}{3}$. The capacitor then begins to accumulate charge again via i_m.

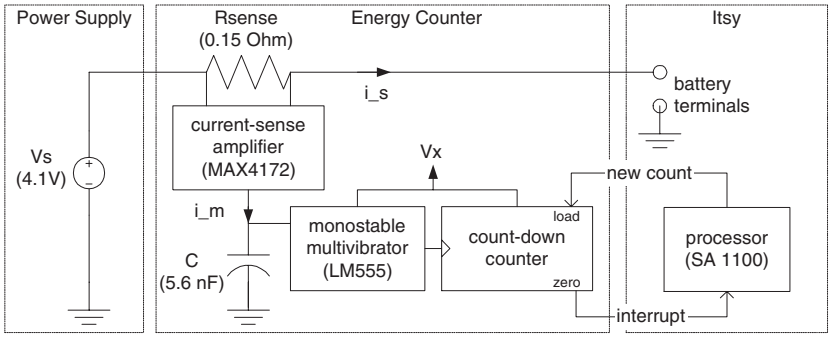

Fig. 1. The Itsy energy-driven statistical sampling prototype. The energy counter is interposed between the power supply and the Itsy pocket computer. Voltage V_x is generated by the Itsy and equals 3.04 V.

Thus, each pulse P indicates that the capacitor (with capacitance C Farads) has accumulated $Q_c = C \times \frac{V_x}{3}$ Coloumbs, during which time the Itsy has consumed $Q_i = \frac{Q_c}{\alpha}$ Coloumbs. Since the voltage powering the Itsy stays constant at V_s, and the voltage drop across the resistor is small (as further discussed in Section 2.3), the energy consumed by the Itsy during this time is approximately $V_s \times Q_i$. In our implementation, this quantity, which we refer to as the *minimum energy quanta*, is approximately 450 μJ.

Each pulse decrements the value of the count-down counter. When this counter's value reaches zero, the counter generates an interrupt request by asserting a signal that is connected to one of the Itsy's general purpose I/O lines (GPIOs). The value of the counter is reset by software, as described in the next section. Thus, the energy counter allows the amount of energy consumption that will generate an interrupt to be varied dynamically. For the experiments reported in this paper, we set the counter value to 1, and hence, our energy quanta is approximately 900 μJ.

2.2 Software

The software for energy-driven statistical sampling is divided into three components – (i) a kernel-level device driver, which includes an interrupt service routine, (ii) a user-level data-collection daemon, and (iii) user-level data-processing tools. This software has been written to run on top of Linux (version 2.0.30), the operating system run by the Itsy. Calls to the device driver control profiling (e.g. turn it on or off). When profiling is turned on, the interrupt service routine handles interrupts raised via the above noted GPIO line. On each interrupt, this routine performs two tasks. First, it records a sample in a circular buffer. A sample consists of the identity of the process, the software module (executable or dynamically linked library), and the instruction within that module that was interrupted. Second, before the routine ends, it resets the value of the count-down counter discussed above. (Calls to the device driver can designate a single reset value, or a small range of values from which the interrupt service routine can randomly select a value.)

The user-level data-collection daemon issues calls to the device driver to obtain sample data, and writes the sample data to the sample data files. At any subsequent time, these files can be processed using the user-level data-processing tools to produce human-readable energy profiles.

2.3 Error Analysis and Validation

In this section, we discuss the sources of error inherent in our energy-driven sampling prototype. Broadly, the sources of error can be classified into two categories: (i) energy measurement related, and (ii) attribution and analysis related. The rest of this section discusses each of these in detail.

Measurement-related Errors

Energy-driven sampling operates on the assumption that each interrupt signifies that the Itsy has consumed a fixed amount of energy. Since our energy counter integrates only the current being drawn by the Itsy, an equivalent assumption is that each interrupt signifies that a fixed amount of charge has been consumed.

The degree to which this assumption holds depends on the degree to which the Itsy's battery-terminal voltage varies. As this voltage drops, the Itsy's power consumption also drops owing to improved voltage-regulation efficiency. This decrease in power consumption is not sufficient, however, to offset the decrease in voltage, and thus, the current drawn by the Itsy increases. Yet, in our implementation, which uses a bench power supply, this supply-voltage dependence problem contributes no significant error. First, we use a well-regulated bench supply, and second, for our benchmarks, the maximum voltage drop across the sense resistor was 52 mV, a value too small to significantly change the current-power relationship for the Itsy. If, on the other hand, the Itsy were powered by its battery, our tool would require an additional mechanism that accounts for the current-power relationship. One possibility would be to characterize this relationship, and, using periodic sampling of the battery voltage, increase the value written into the count-down counter as the battery voltage decreases.

A second source of measurement-related error is the inherent non-ideal characteristics of electrical components. Of particular note is the time required for the multivibrator to discharge the capacitor, which we determined empirically to be approximately 290 ns. For our benchmarks, the total time spent during such discharges of the capacitor represents less than 0.1% of the total time. The other component-related errors are also negligible due to careful selection of components and operating ranges.

A third source of measurement-related error is the impact of the power consumed by the energy counter on the resulting sample distribution. This error is dominated by the heat dissipated by the 0.15 Ohm resistor, which, for our benchmarks, was at most 17 nW. To put this number in perspective, the maximum power consumed by our benchmarks is 1.4W.

Finally, a fourth source of error is the degree to which the sampling software perturbs the resulting energy profiles. All the sampling software except the interrupt service routine is profiled. Therefore, the percentage of samples attributed to this software serves as an estimate of the error it induces. For each of our benchmarks, this percentage

Table 1. Estimates of the impact of sampling software on energy profiles.

Benchmark	ISR overhead (est.) (1)	daemon & driver overhead (2)
cpu-bound-1	0.35%	0.25%
cpu-bound-2	0.49%	0.20%
idle	0.66%	0.01%
mpeg	1.32%	0.50%
synth	1.62%	0.11%
voice	0.77%	0.15%
midi	1.08%	0.02%
wave	0.74%	0.01%
zip	1.13%	0.29%
ftp	0.92%	1.49%
qpe1	1.54%	0.14%
qpe2	1.58%	0.17%
qpe3	1.48%	0.20%
edemo	1.45%	0.17%

is given in column 2 of Table 1. (The benchmarks are described later in Table 2.) Although the interrupt service routine cannot be profiled, we can measure the time spent in the routine. We then estimate the number of samples that would have been attributed to the routine by assuming that the routine's power consumption is approximately the same as our `zip` benchmark, which (like the interrupt service routine) is memory-intensive, does not include any idling, and does not use any devices. This estimate, as a percentage of the total number of samples collected, is given in column 1. That the percentages in columns 1 and 2 are so low indicates that the profiling software did not incur significant error.

Attribution and Analysis Errors

The other major class of errors is attribution and analysis errors. To accurately attribute a sample to an instruction, it is important to minimize the delay between the following two events: the monostable multivibrator generating a pulse P that will decrement the counter to zero (see Section 2.1), and the interruption of the program in execution so that the interrupt can be serviced. This delay is composed of the time required for the interrupt to be conveyed to the processor core, and the time required for the processor to begin servicing the interrupt. For our prototype, the first component is on the order of nano-seconds, and is thus a small fraction of the time between interrupts, which is on the order of milli-seconds. Given that the power consumed during such small time intervals does not vary significantly, the first component's impact is a marginal variation in the energy quanta.

Regarding the second component, deferred interrupt handling could incur delay in some specific situations (because the processor is handling a higher priority interrupt, for example), but such situations occur infrequently. In addition, the Itsy pocket computer uses a StrongARM SA1100 processor. One characteristic of this processor is that,

before servicing a trap or exception, the processor first completes execution of all instructions in the *decode* and later pipeline stages. Thus, when the energy-counter interrupt is processed, the sample will be attributed to the instruction that was fetched *after* the instruction that was in execution at the time that the interrupt was delivered to the core. Detailed information about instruction delays, pipeline scheduling rules and interrupt handling on the SA1100 processor could be leveraged to adjust for this processor-induced skew in attribution. Without this information, however, this skew will be common to all interrupt-driven statistical sampling tools for the SA1100 processor (including the ones discussed in this paper).

A second source of error arises if phases of the application being profiled are synchronized exactly with the discharging of the capacitor. Although none of our benchmarks exposed this error, our prototype allows synchronization errors to be avoided by dynamically varying the energy quanta after which an interrupt will be generated (as described in Section 2.2).

Finally, a third source of error concerns sensitivity to the sampling rate. The accuracy with which a sample distribution reflects the true allocation of energy consumption for an application is proportional to the square root of the number of samples [5]. However, larger number of samples can lead to greater processing overhead and increased profiler intrusiveness. For our benchmarks, our default parameters resulted in the collection of 12,000 to 125,000 samples at 130Hz-1100Hz sampling frequencies. Sensitivity experiments with longer execution times and higher sampling frequencies indicated that this setup did not have appreciable errors.

3 Comparison of Sampling Approaches

This section presents a study comparing energy-driven statistical sampling to existing time-driven statistical sampling approaches. We begin by describing our experimental methodology, with Section 3.1 describing our time-driven statistical sampling prototypes, and Section 3.2 describing our 14 benchmarks. Section 3.3 then presents the results of our comparison study.

3.1 Time-driven Statistical Sampling Prototypes

We compared our energy-driven statistical sampling tool with two versions of time-driven statistical sampling, a simple time profiler and our implementation of the PowerScope approach [7]. The simple time profiler requires no hardware support and is the existing tool available with the Itsy public distribution [9]. Both our energy-driven sampler and our PowerScope implementation use the same software, with minimal modifications.

Figure 2 depicts our PowerScope prototype. Our prototype differs from the original PowerScope implementation in that we use a data acquisition system rather than a digital meter to measure the instantaneous current of the system. The data acquisition system generates a clock that drives both time-driven sampling on the Itsy and the collection of instantaneous current measurements. The current measurements are correlated off-line with the samples recorded by the interrupt service routine. Since the

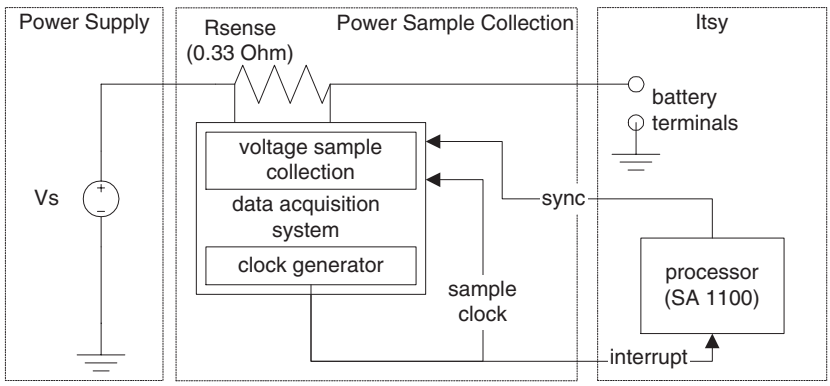

Fig. 2. Our implementation of PowerScope. The data acquisition system has its own power supply (unshown).

battery-terminal voltage of the Itsy varied by approximately 2%, the current measurements were directly proportional to the Itsy's instantaneous power consumption. Energy profiles were generated by weighting each sample by its correlated current measurement, under the assumption that the instantaneous power consumption of the system sufficiently approximates the average power consumption since the last sample.

Error Analysis for Prototypes

For the PowerScope prototype, calibration experiments indicate that the current measurement error introduced by the data acquisition system is significantly lower than 1%. The data acquisition system is powered by a separate power supply to minimize other perturbances.[3] To avoid sample-collection and application synchronization, our simple time profiler jitters the time between successive interrupts (by 1/16 of the base frequency). Note that the instruction-level sample-attribution skew discussed in the Section 2.3 is intrinsic to the SA1100 processor, and therefore common across all our prototypes.

3.2 Benchmarks

Table 2 describes the fourteen benchmarks we studied, their execution times, and their average sampling frequencies for our energy-driven sampler. The benchmarks were chosen to cover a wide range of activities. They include two contrived CPU-intensive benchmarks (`cpu-bound1` and `cpu-bound2`), a variety of media processing benchmarks (`mpeg`, `synth`, `voice`, `midi`, and `wave`), file handling and file transfer benchmarks (`zip` and `ftp`), and a few complex benchmarks (`qpe` and `edemo`) that include a combination of the functionality of several of the other benchmarks.

[3] We also implemented an *online* version of PowerScope using the hardware provided by the Itsy for measuring its own instantaneous current consumption. However, the power consumed by these components was sufficiently large to cause significant perturbances in the profiles.

Table 2. Benchmarks studied.

Benchmark	Description	Time (sec)	Avg samp freq (Hz)
cpu-bound-1	Two interleaved loop-based compute-intensive (i.e., addition) procedures	193	441
cpu-bound-2	Six instances of a loop-based compute-intensive procedure interleaved with sleep periods	444	282
idle	Itsy in idle mode	299	130
mpeg	Two MPEG movies (bigtoy.mpg and swars2.mpg) played sequentially for six iterations	171	1113
synth	Speech synthesis of a 6KB text document (samples scaled by 0.25)	375	592
voice	Recording dictation of two paragraphs	41	297
midi	MIDI file (3-spain.mid) played to completion using Timidity player	342	411
wave	Itsy.wav sound file played four times	141	298
zip	Three instances of tarring, zipping, and removing a 573KB directory	30	782
ftp	FTP' ing of a 573KB file	55	291
qpe1	Windowing environment trace involving operations on the calculator program, the tux toy program, and the window canvas program	245	493
qpe2	Windowing environment trace involving operations on the datebook program (read, edit, delete, add)	240	570
qpe3	Windowing environment trace involving typing a few paragraphs on the keyboard	259	668
edemo	Itsy E-mail demo, which uses command-and-control speech recognition to process e-mail. Test included voice-directed and button-based navigation, speech synthesis of several messages, and recording audio of a response.	344	524

To ensure repeatability, the experiments with `qpe` – the only benchmark that needed a lot of user input through the touch screen – were automated by modifying the touch-screen driver to play back pre-recorded traces. In addition, most of the benchmarks were chosen to include multiple instances of an application, or to take enough time to ensure a large number of samples. To ensure consistency, we used a single Itsy for all our experiments. In addition, for each benchmark, we set the sampling frequency of the time-driven approaches to the average sampling frequency with energy-driven sampling, thereby ensuring that all approaches obtain a similar number of samples. However, for two benchmarks (`mpeg` and `qpe`), matching frequencies caused the PowerScope prototype to generate trace files that exceeded the available storage on the Itsy. Therefore, in these two cases, we used the highest frequency possible that did not exceed the available storage – 500 Hz for `mpeg` and 100 Hz for `qpe`. The running time and periodicity of these benchmarks was sufficient to ensure that the lower sampling rate did not have a significant impact on our results.

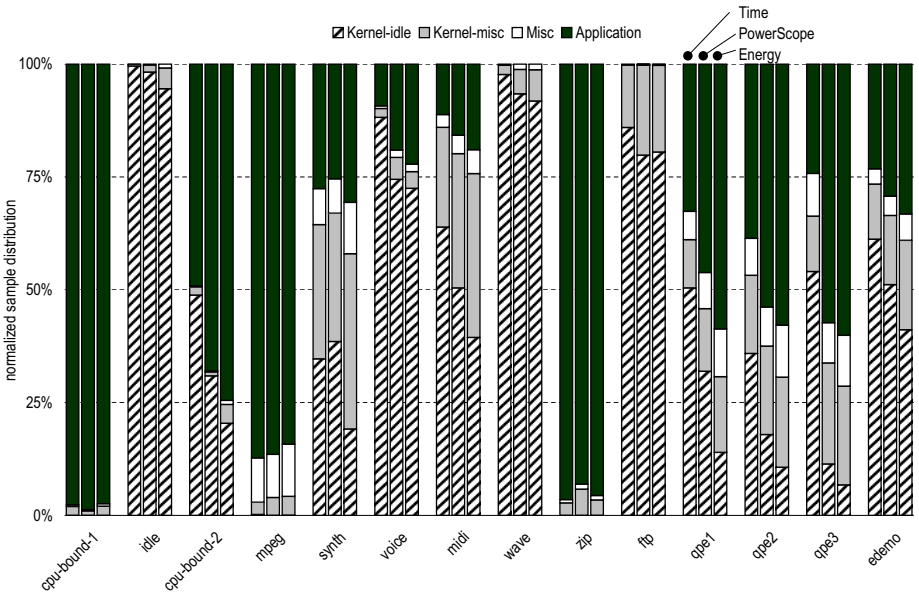

Fig. 3. Module-level comparison of time and energy profiles.

3.3 Experimental Results

Figure 3 summarizes the profiles for the fourteen benchmarks. For each benchmark, the bar on the left (Time) summarizes the profile generated by the simple time profiler, the bar in the middle (PowerScope) summarizes the profile generated by the PowerScope prototype, and the bar on the right (Energy) summarizes the profile generated by the energy-driven sampling tool. Each profile is divided into four components – the application modules (Application), the kernel-idle loop (Kernel-idle), other kernel modules (Kernel-misc), and miscellaneous library modules (Misc).

Prodfile Differences

Figure 3 shows that the different tools often produce significantly different results. Eight of the fourteen benchmarks in our study show substantial differences between the time and energy profiles. While the differences between the energy profiles generated by our PowerScope prototype and our energy-driven sampling tool are smaller, there are still substantial differences for six of the fourteen benchmarks.

The differences between profiles are particularly evident for benchmarks that cycle through *multiple power states*. In our benchmarks, the most dominant example of such multiple power states is the difference between the power consumed while executing the kernel-idle loop and any other code. For example, the synthetic `cpu-bound-2` benchmark consumes about 100mW when in the idle loop, but about 400mW when in the cpu-intensive portions of the benchmark. Figure 4 illustrates this difference. This

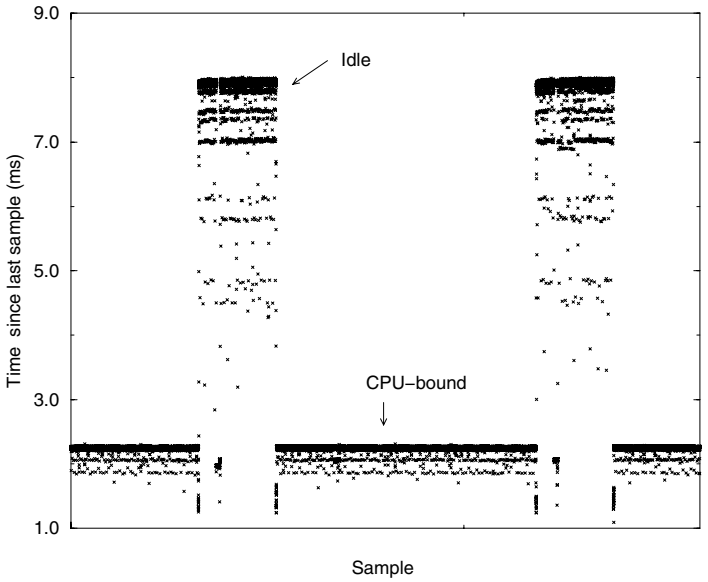

Fig. 4. Variation in time between interrupts for the `cpu-bound-2` benchmark

figure plots the time since the last sample for the sequence of samples obtained by our energy-driven sampler. Since one energy quanta was consumed during each sampling interval, larger time values correspond to periods of lower power consumption.[4] Notice that the time values while the processor is idle are about four times those of when the processor is busy.

The simple time profiler consistently over-estimates the energy consumption of the kernel-idle loop. PowerScope detects some of the difference between the idle and non-idle modes, but still attributes too much energy to the kernel-idle loop. This discrepancy arises because the current drawn by the Itsy while idle is not constant. Rather, due to background system activity (e.g. real-time clock updates and daemon activations), there are occasional pronounced current peaks of short duration. This is illustrated in Figure 4 by the multiple bands of sampling intervals associated with the idle periods.

[4] Thus, since the average power consumption during each sampling interval can be determined by dividing the energy quanta by the time between successive samples, energy-driven sampling may also be used to obtain a time-sequenced set of power measurements. Similarly, with energy-driven sampling, the relative power consumption between two tasks can be compared simply by comparing the average sampling rate during the two tasks. Of course, such information could also be obtained using other means (e.g. digital meters or a data acquisition system).

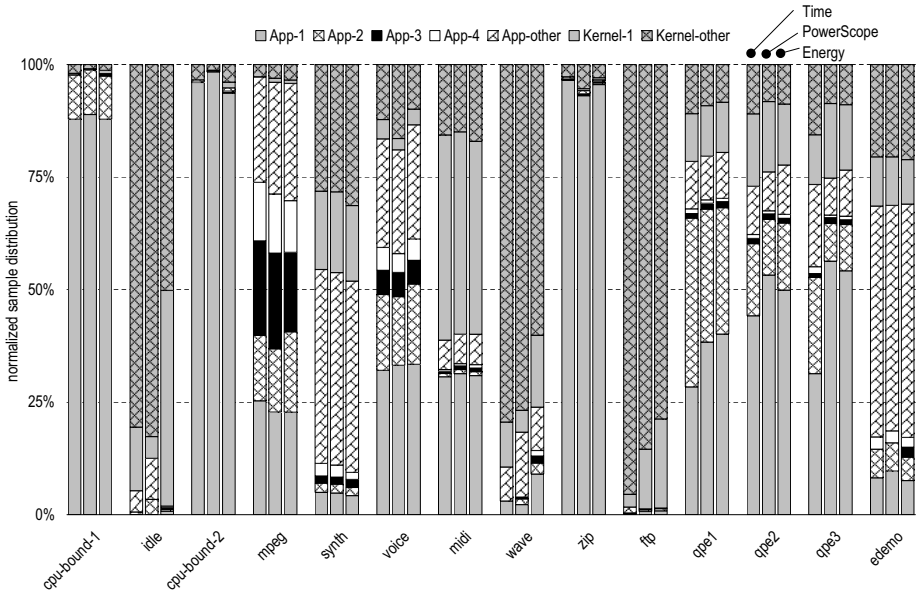

Fig. 5. Procedure-level comparison of time and energy profiles that exclude idle-routine samples.

Since PowerScope only samples instantaneous power, it would require a much higher sampling rate to capture such transient effects accurately.

To investigate the impact of power states other than idle mode, Figure 5 shows profiles for the fourteen benchmarks that were generated by ignoring samples in the kernel-idle loop. As in Figure 3, for each benchmark, the bars on the left, middle and right summarize the profile generated by our simple time profiler, our PowerScope prototype and our energy-driven sampling tool, respectively. In addition, to compare the profiles generated by the different approaches at a finer granularity, each profile is now divided into seven categories. The profiles for each benchmark distinguish the four application procedures (App-1 to App-4) and the (non-idle) kernel procedure (Kernel-1) that received the greatest number of samples using our energy-driven sampler.

Figure 5 reveals that there continue to be differences between the profiles even after idle time is factored out. However, the differences are both smaller and less common. Indeed, aside from the `qpe1` and `qpe3` benchmarks, it is not apparent whether the differences between time and energy profiles at this granularity would be significant to a programmer who would use a profile to steer development effort focused on reducing energy consumption. Note that the profiles for the `idle`, `wave`, and `ftp` benchmarks also show variations, but the significance of these variations is questionable since 80% or more of the samples for these benchmarks were attributed to the kernel-idle loop, as shown in Figure 3.

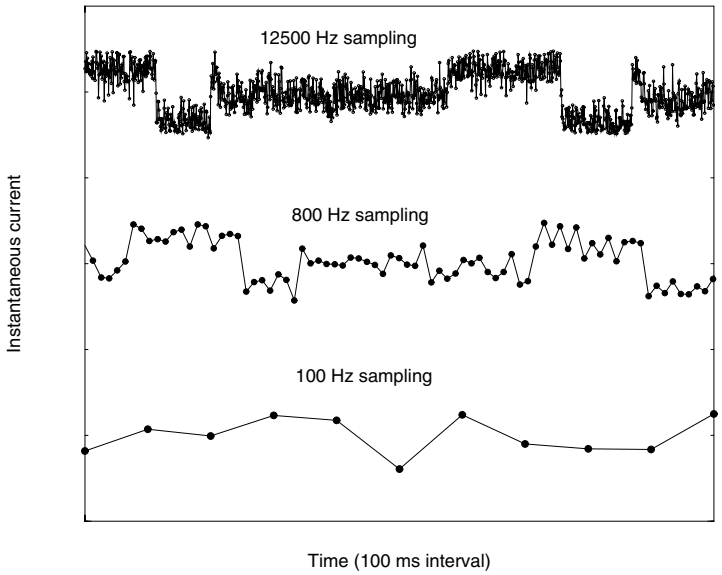

Fig. 6. Variation in instantaneous current samples during the `mpeg` benchmark.

Profile Resolution

Considering the variations we observed in instantaneous power measurements, we were surprised by the absence of more significant differences in the profiles obtained by our energy-driven sampler and our PowerScope prototype. For example, Figure 6 shows instantaneous current readings obtained by sampling at three different frequencies (100 Hz, 800Hz, and 12.5 KHz) during the `mpeg` benchmark. Recall that the PowerScope approach assumes that the instantaneous power consumption associated with each sample approximates average power consumption since the prior sample. Comparing each sample value at 100 or 800 Hz with the average across 12.5 KHz sample values since the prior 100 or 800 Hz sample reveals more than a few instances in which this assumption does not hold. This observation suggests that the profile generated by the PowerScope approach could contain substantial inaccuracies, a hypothesis that explains the approach's over-attribution of energy to the kernel-idle loop. Interestingly, however, when the samples attributed to this loop are ignored (Figure 5), the profiles generated by the two approaches do not differ substantially.

To help understand this somewhat surprising result, we compared the instruction-level profiles produced by our energy-driven sampler and our PowerScope prototype. Figure 7 illustrates this comparison for the `mpeg` benchmark. Each point on this graph corresponds to a single instruction. The location of a point is determined by the percent-

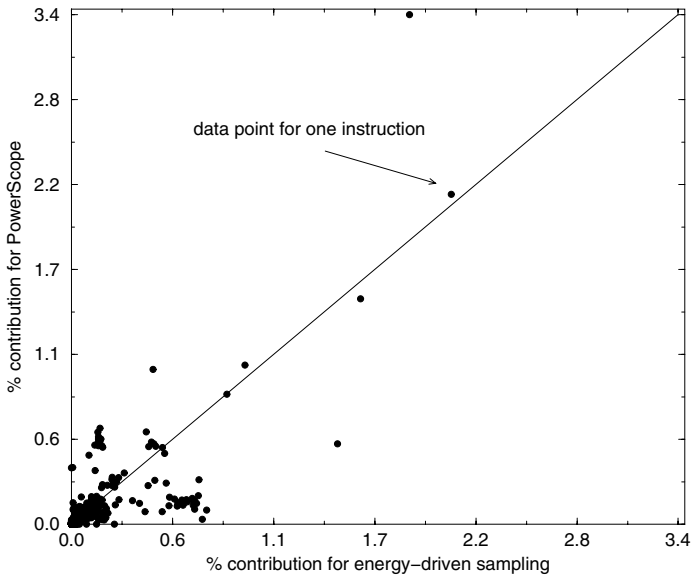

Fig. 7. Differences in instruction-level profiles for the `mpeg` benchmark.

age of samples attributed to that instruction by energy-driven sampling (X-axis value) and the PowerScope approach (Y-axis value). Hence, if the instruction-level profiles of the two approaches were nearly identical, all data points would lie on or near the x=y diagonal. The presence, in Figure 7, of a significant number of outliers indicates that there are indeed substantial differences in where the two approaches attribute energy consumption at an instruction level.

It appears that instruction-level differences simply average out with increasing profile granularity. To validate this hypothesis, we estimated how the difference between the profiles produced by these two approaches changes as the granularity of the profiles increases. In particular, within each instruction-level profile, we distributed the samples attributed to each instruction I uniformly across the instructions with PC addresses within n of I. Then, for different values of n, we calculated the difference between (redistributed) profiles as the sum of the absolute differences between the number of samples attributed to each instruction. This method ignores control flow, but provides a first order estimate of the impact of granularity. In support of our hypothesis, we observed that, for increasing values of n between 0 and 16, the difference between the profiles decreased. Furthermore, for most of the benchmarks, there was a 55% to 75% drop in this difference between $n = 0$ and $n = 16$.

Impact of Other Power States

As noted already, in our experiments, the most significant variations in the profiles were due to multiple power states caused by the kernel-idle component. In this section, we examine two other important power states: those due to cache misses, and those due to frequency/voltage scaling.

Empirical data [12] indicates that the SA1100 may not exhibit multiple power states at the processor level for most instructions. However, cache misses and the resulting memory accesses may produce another power state that may lead to dominant differences between the different tools.

The experiments reported in this paper were obtained with the StrongARM SA1100 run at 133 MHz with clock switching enabled. Clock switching allows the core clock frequency to drop to the bus speed until the required data is read from memory. The reduction in the core power offsets the increased power for the memory access leading to comparable power consumption for cache hits and cache misses (less than 10% variation [8]). However, we performed several experiments with a synthetic cache-miss workload at a higher frequency (206 MHz) where cache misses took twice as much power as cache hits [8]. Our results showed marked differences between time and energy profiles, with the differences proportional to the cache miss ratios.

Additionally, the SA1100 is a fairly simple processor with a limited number of power states [12]. More complex processors are likely to have more instructions that produce additional power states, which would increase the differences between profiling approaches.

Frequency and voltage scaling, like cache misses, also offer the potential for additional power states that can be cycled through rapidly. The impact that frequency and voltage scaling may have is suggested by the following two experiments. In the first experiment, we obtained time and energy profiles for a workload comprised of running a long benchmark twice, first at a clock frequency of 206 MHz, then at a clock frequency of 59 MHz. In the second experiment, we obtained time and energy profiles for a workload comprised of running a short benchmark six times, each time with a different clock frequency (206 MHz, 176 MHz, 148 MHz, 118 MHz, 89 MHz and 59 MHz). In both cases, the energy profile was obtained using our energy-driven sampler (rather than our PowerScope prototype). Table 3 summarizes the results of these two experiments. While these workloads are synthetic, the results indicate the inappropriateness of time profiles for estimating the energy consumption of applications that exploit frequency scaling. Moreover, since PowerScope assumes that instantaneous power consumption approximates the average power consumption between successive samples, rapid cycling through multiple energy states may increase the difference between energy-driven sampling and the PowerScope approach.

3.4 Observations

Based on our results, we can make several observations.

- Simple time profiles do not accurately reflect energy consumption in many cases. Such inaccuracies are particularly pronounced when software cycles through multiple power states (e.g., idle mode versus CPU-busy).

Table 3. Impact of frequency scaling.

Module	Time (%)	Energy (%)
App-206	22.01	41.13
App-59	77.44	57.50
Other	0.56	1.37

Module	Time (%)	Energy (%)
App-59	31.36	20.52
App-89	20.84	17.56
App-118	15.67	16.32
App-148	12.47	15.57
App-176	10.33	14.92
App-206	8.85	14.56
Other	0.46	0.54

- Energy-driven profiles give better instruction-level resolution than previously proposed time-driven power sampling approaches (e.g., our PowerScope prototype) at the same sampling frequency. Furthermore, when multiple power states are exercised, they may provide greater accuracy with lower sampling frequencies.
- However, on simple handheld systems which do not exercise multiple power states other than idle mode, time-based profiling approaches may sufficiently approximate energy profiling for the purposes of assisting programmers – without requiring any additional hardware support.

4 Comparison with Activation-model Approaches

Prior work has also explored approaches to exposing more information about a system's energy consumption that are not based on statistical sampling. These approaches, which we refer to as activation-model approaches, require two components: (1) energy-use estimates for specific system activities, such as executing a specific type of instruction, idling the processor, accessing memory or a disk, and sending or receiving messages over a network; and (2) activation counts for each activity, which are obtained by counting the number of times each activity occurs when an application runs. Energy consumption for the application can then be obtained by multiplying the activation counts by the energy cost per activation.

Activation-model approaches differ widely in how they obtain activation counts. They can be classified as counting-based, sampling-based, or simulation-based approaches. With counting-based approaches, activation counts are obtained by either running instrumented versions of applications (e.g., Millywatt [4]), or running (unmodified) applications on an operating system that is modified to leverage hardware event counters (e.g., Joule Watcher [2]). A sampling-based approach is put forth by PowerMeasure/StateProfiler [10], which periodically samples the system's state to estimate some activation counts. Finally, with simulation-based approaches (e.g., SimplePower [13] and Wattch [3]), a specially-designed simulator is used to collect the activations of interest for an application.

One advantage of activation-model approaches is that they can leverage activation counts for background activities to more easily attribute energy consumed asynchronously. In contrast, both the PowerScope and energy-driven samplers we evaluate in this paper attribute the energy consumed by all system activity to the software modules that are executing while this activity is on-going. For example, should a software

module initiate DMA, the samplers would attribute the energy consumed by the DMA engine to all the software modules that execute while the engine is busy. Thus, while this approach to attributing energy consumption gives programmers insight into system-level energy consumption, it does not identify the modules that are actually responsible for such background activity. We are currently exploring ways to augment statistical sampling to enable more sophisticated attribution. Section 5 briefly describes one of the approaches we are considering.

The main disadvantage of activation-model approaches is the difficulty of ensuring their accuracy. In particular, the accuracy of these approaches depends on tracking all important activation counts, and on obtaining accurate energy-use estimates for all important system states - often through detailed simulation or application instrumentation. Obtaining such estimates is non-trivial for a single platform, let alone many platforms. In contrast, our tool requires neither prior intuition about what activities may be of interest nor energy-use estimates for specific activities.

5 Summary and Future Work

While the issues with designing applications to reduce execution time are fairly well understood, a similar understanding about how to design applications to reduce their energy consumption is lacking. This paper presented a new approach, *energy-driven statistical sampling*, to exposing information about energy consumption. Tools developed with this approach can help the software designer both reason about the energy impact of software design decisions and identify application energy hot spots.

Energy-driven statistical sampling uses a small amount of hardware to trigger an interrupt at pre-defined quanta of energy consumption. The interrupt is used to collect information about the program currently executing, and the information thus collected is processed off-line to generate an energy profile of where energy was spent during the program's execution. We have developed a prototype of this approach for the Itsy mobile computing platform and our studies on the prototype indicate that energy-driven statistical sampling can provide an accurate system-level software energy profile with very little overhead.

We compare energy-driven statistical sampling to two time-driven statistical sampling approaches by comparing the profiles generated by these approaches for 14 benchmarks programs. Our results show that energy-driven statistical sampling can provide better instruction-level resolution and greater accuracy than existing time-driven statistical sampling approaches. In particular, there are often significant differences between the profiles generated by energy- and time-driven statistical sampling when the workload cycles through multiple power states. On simple handheld systems, many applications may exercise only a single power state other than idle mode, in which case time profiling may sufficiently approximate energy profiling for the purpose of assisting programmers. However, preliminary investigations indicate that emerging functionality, like frequency and voltage scaling, will increase the differences between time and energy profiles, and therefore the benefit of energy-driven statistical sampling.

As part of our ongoing efforts, we plan to extend our work to account for background power usage. One solution we are exploring is to augment the energy-counting

hardware of a system to include software-readable counters that track the amount of energy consumed by background energy consumers (e.g., the backlight, a wireless radio). These counters could then be used by our interrupt service routine to assign fractions of the sample (based on energy usage) to system energy-use categories (e.g., backlight on, wireless active). In this way, a programmer may identify major energy consumers at the hardware level and, with further investigation, the software modules responsible.

We are also repeating this comparison study in the PocketPC environment on the iPAQ pocket computer, and are considering doing so on systems that support frequency and voltage scaling.

Acknowledgment

A number of people assisted us with the research described in this paper and with the larger effort of which it is a part. We thank Deborah Wallach and Marc Viredaz for helping us with the Itsy Pocket Computer hardware and software and Wayne Mack for helping with soldering and component assembly. Also, for supporting our research in numerous ways, we thank Ken Nicholas, Alan Eustace, Jim Mann, Ramakrishna Anne, George Bold, Scott Briggs, Tim Kamrath, and Tu Nguyen. Finally, we thank the anonymous reviewers for their comments.

References

1. J. Anderson, L. Berc, J. Dean, S. Ghemawat, M. Henzinger, S. Leung, D. Sites, M. Vandevoorde, C. Waldspurger, and W. Weihl. Continuous profiling: where have all the cycles gone. In *Proceedings of the 16th Symposium on Operating Systems Principles*, October 1997.
2. F. Bellosa. The benefits of event-driven energy accounting in power sensitive systems. In *Proceedings of the 9th ACM SIGOPS European Workshop*, September 2000.
3. D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *Proceedings of the 27th International Symposium on Computer Architecture (ISCA)*, June 2000.
4. T. L. Cignetti, K. Komarov, and C. Ellis. Energy estimation tools for the Palm. In *Proceedings of the ACM MSWiM'2000: Modeling, Analysis and Simulation of Wireless and Mobile Systems*, August 2000.
5. J. Dean, J. E. Hicks, C. Waldspurger, B. Weihl, and George Chrysos. ProfileMe: Hardware support for instruction-level profiling on out-of-order processors. *In Proceedings of the 30th Annual International Symposium on Microarchitecture*, December 1997.
6. X. Zhang et al. Operating system support for automated profiling and optimization. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles*, October 1997.
7. J. Flinn and M. Satyanarayanan. PowerScope: A tool for profiling the energy usage of mobile applications. In *Proceedings of the Workshop on Mobile Computing Systems and Applications (WMCSA)*, pages 2–10, February 1999.
8. Jason Flinn, Keith I. Farkas, and Jennifer Anderson. Power and energy characterization of the itsy pocket computer (version 1.5). Technical Report Technical Note TN-56, Compaq Western Research Laboratory, February 2000.
9. W. R. Hamburgen, D. A. Wallach, M. A. Viredaz, L. S. Brakmo, C. A. Waldspurger, J. F. Bartlett, T. Mann, and K. I. Farkas. Itsy: Stretching the bounds of mobile computing. *IEEE Computer*, 34(4), April 2001.

10. J. Lorch and A. J. Smith. Energy consumption of Apple Macintosh computers. *IEEE Micro Magazine*, 18(6), November/December 1998.
11. T. Simunic, L. Benini, and G. De Micheli. Energy-efficient design of battery-powered embedded systems. In *Proceedings of the International Symposium on Low-Power Electronics and Design '98*, June 1998.
12. Amit Sinha and Anantha P. Chandrakasan. Jouletrack - a web based tool for software energy profiling. In *Design Automation Conference (DAC 2001)*, June 2001.
13. W. Ye, N. Vijaykrishan, M. Kandemir, and M. J. Irwin. The design and use of SimplePower: A cycle-accurate energy estimation tool. In *Proceedings of the Design Automation Conference*, June 2000.