A Run-Time, Feedback-Based Energy Estimation Model For Embedded Devices

Selim Gurun and Chandra Krintz
Dept. of Computer Science, UC Santa Barbara
Santa Barbara, CA, 93106
gurun@cs.ucsb.edu,ckrintz@cs.ucsb.edu

ABSTRACT

We present an adaptive, feedback-based, energy estimation model for battery-powered embedded devices such as sensor network gateways and hand-held computers. Our technique maps hardware and software counters to energy consumption values using a set of first order, linear regression equations. Our system is novel in that it *combines online and offline techniques* to enable runtime power prediction. Our system employs an offline instantiated model that it continuously updates using feedback from a readily available battery monitor within the device.

We empirically evaluate our model and detail its robustness, accuracy, and computational cost. We also analyze the stability of the model in the presence of feedback errors. We demonstrate that our approach can achieve an error rate of 1% (extant techniques: 2.6% to 4%) for computationally bound tasks and 6.6% (extant techniques: 11%) for communication bound tasks.

Categories and Subject Descriptors: D.4.8 [Operating Systems]: Performance – Modeling and prediction, Measurements

General Terms: Measurement, Performance

Keywords: Power and Energy Estimation, Power Modeling, Battery Monitoring Unit, Battery Powered Devices

1. INTRODUCTION

Battery-powered, embedded devices have experienced widespread use in recent years as a result of their growing capability for executing increasingly resource-intensive tasks. These devices, however, are severely limited in energy supply. To enable energy-efficient software, novel techniques are required to characterize accurately and efficiently the energy consumption of the device.

To this end, we propose a power model to provide accurate, fine-grain energy consumption estimations *at runtime* using feedback from a battery monitor. Such battery monitors currently exist on many portable devices such as cell phones and PDAs and measure the amount of charge flowing through the circuit. This paper presents a model that couples coarse-grained battery monitor data with low-level hardware and software performance counters to estimate fine-grain energy consumption dynamically.

Our primary goal is to develop a system that exploits currently

available hardware. We consider two common mobile devices, the HP Ipaq H5550 [5] hand-held computer and the Stargate sensor network gateway. Both devices have an Intel XScale [9] processor, a network interface, and expansion capability via a PCMCIA socket. The most significant difference between the Stargate and the iPAQ is that the former does not have a touch-screen/display.

Our model must also be highly accurate, computationally simple, and responsive (adaptive) to changes in the workload, device, and battery characteristics. High accuracy (i.e. low estimation error) is critical for the efficacy of energy-aware optimization that employ estimations from our model. Moreover, since our model will be used online, *while* the device is being employed by users, it must introduce little overhead and consume few resources (including the battery). Finally, since the way in which the device is used changes over time, our model must adapt to these changes to ensure high accuracy over time.

Prior work [4, 12, 10, 6] presents accurate energy models for the CPU in isolation, that achieve an error rate of 3% to 4%. However, the CPU consumes only a fraction of the total energy [2]. Other devices, such as I/O and system controllers have a significant impact on total energy consumption. Our end-goal is a full-system model that can provide such information to the application at run time. As an initial step in this direction, we present an adaptive model that considers tasks that execute on the RAM drive and perform I/O via a wireless network card.

The contribution of this paper is an adaptive, low cost, highly accurate runtime power estimation model for battery powered embedded devices. The proposed model uses runtime feedback to update model parameters by using a recursive least squares regression technique. We perform an extensive performance and accuracy evaluation of this technique and compare it to existing models. We find that our model achieves an average error rate of 1% for CPU bound tasks and 6.6% for network tasks. We also show that feedback mechanism can actually increase the error rate rather than reducing it, if the model is not well-designed. Our technique can be applied to embedded devices such as HP Ipaqs and Stargate sensor network gateways and used to guide energy-aware optimization.

We first describe our high level design principles, present the features and constraints of our target platform, and discuss our power model. In Section 3, we evaluate various design parameters and show how they effect the model accuracy, efficiency, and robustness. In the final sections we discuss related work and conclude.

2. RUN-TIME FEEDBACK-BASED ENERGY ESTIMATION MODEL

We first present our power model and the components of our power-estimation design. We then describe the battery monitor which provides feedback to the model at runtime.

2.1 Power Models

Prior studies model energy consumption of CPU and memory using static, first-order, linear regression equations [4, 6, 13]. Unfortunately, static models suffer from a high error rate when constituent control signals, such as the CPU clock rate, changes over time. [6] shows that a static CPU model must be redesigned for each supported clock frequency. Similarly, I/O usage can change energy consumption significantly over time [2]. As a result, we have designed our model to dynamically adapt to such changes in an effort to improve model accuracy.

In this work we focus on computation and communication behaviors. We develop our models following prior work [4, 6, 8] using hardware performance monitors (HPMs). The HPM hardware provides feedback about program execution behavior by monitoring various system events, such as data cache accesses, CPU cycles, etc. The XScale CPU that we model has 2 event counters.

Our computation model estimates the energy consumption of tasks that execute without any communication or any significant access to persistent storage. This model includes three parameters:

$$E(Joules) = \alpha_0 + \alpha_1 x_1 + \alpha_2 x_2 \qquad (1)$$

where x_i's are core clock cycles and data stalls, respectively. We compute the parameter weights, α_i, offline and adaptively fine-tune them at runtime. We estimate energy consumption for fixed periods of program's execution, which we refer to as intervals.

Our communication model uses two *software performance counters* that we developed to characterize communication patterns: transmit bytes (B_{tx}), receive bytes (B_{rx}) and one hardware counter, core clock cycles. We do not include other HPM events to reduce the cost of RLS-ED iterations. Our communication model is:

$$E_n(Joules) = \alpha_1 x_1 + \beta_1 B_{tx} + \beta_2 B_{rx} + K \qquad (2)$$

Here, α_1 is the weight of core clock cycles and β's are weights of transmit and receive bytes. At present, we did not incorporate low level card state (idle, etc) information to our model, as we favor a simpler model at the expense of a potentially higher error rate.

Due to its stability, robustness, adaptivity and modest computational demand, we use *recursive least squares linear regression with exponential decay* [18] (i.e. *RLS-ED*) to compute the model parameters. The RLS-ED is a recursive implementation of the well-known least squares linear regression. Using a decay factor, it exponentially reduces the weight of the oldest measurements. With the measurements u_k at time k, and the decay factor γ, RLS-ED weights the measurements using:

$$u_k + \gamma u_(k-1) + \gamma^2 u_(k-2) + ... + \gamma^k u_0$$

The γ adjusts the adaptiveness of the algorithm ($\gamma \leq 1.0$). A smaller γ means the model is more responsive to changes in the input data but less resilient to noise. In Section 3, we discuss the effect of γ on regression accuracy and stability.

2.2 Power Estimation Framework

Our online power estimation framework consists of three components: a runtime profiler, an offline profiler, and a power estimator. The runtime profiler polls the battery monitor in the device and accumulates values from the software and hardware counters periodically. After each period, the profiler runs RLS-ED to iteratively update the parameter weights. Internally, RLS-ED maintains a matrix of size $n \times n$ (n is equal to the number of model parameters) to retain the state information between each iteration.

Each RLS-ED iteration involves eight matrix multiplications, each of which requires approximately one hundred floating point operations when $n = 2$. Even though this may not be a significant cost

on high-end machines, these floating point operations can consume significant resources on many resource-constrained platforms. To reduce this cost, we explore policies to reduce RLS-ED iteration frequency, in the next section. In addition, since the asymptotic complexity of the algorithm is $O(n^3)$, we must also keep the number of model parameters small, to keep n small and the computational cost of the algorithm low. We discuss RLS-ED execution period and parameter selection policies later in Section 3.

Depending on the application state, the *energy estimator* chooses between Equation 1 and Equation 2 to predict the energy consumption for each interval. Each estimation in our model requires fewer than 10 floating point operations.

The *offline profiler* is the only optional component of our system. As the RLS-ED algorithm is recursive, it requires an initial state to start its iterations. Without the existence of an offline profiler, the error rates can be high until the algorithm reaches a stable state. The offline profiler reduces this initial warm-up time by generating an initial state using profiled data.

2.3 Battery Monitoring Unit

The battery monitoring unit (BMU) is a device that continuously monitors battery voltage, temperature, and current that is flowing into and out of the battery. BMUs are commonly available in battery-powered devices to provide users and system software with feedback as to the remaining battery life. Our runtime profiler uses this circuit to measure energy consumption and to update the model parameters recursively.

The BMU that we describe [7] is an integrated part of the many other embedded devices including HP Ipaqs and the Stargate that we use in this study. When the system is on battery power, the BMU samples the current flow 128 times per second and stores the result in an internal register. At run time, the energy consumption from time t_1 to time t_2 can be computed by reading this register. More specifically, let (v_1, ac_1) and (v_2, ac_2) be voltage and **accumulated current** readings at time t_1 and t_2. The energy consumption at $[t_1, t_2]$ is:

$$E = (v_1 + v_2)/2 \times (ac_1 - ac_2) \times 3600 \text{ sec/hours}$$

This equation does not include time since ac is the accumulated current and not the average current. The multiplier, 3600 sec/hours, converts the result to microjoules.

This equation however, is severely limited by the precision of A/D converter which is 12 bits in our BMU. Thus the difference between the real and measured values can be up to 4.88 mV for voltage and 0.25 mAh for accumulated current. For this reason, the BMU cannot be used alone for energy estimation – we must couple it with a more accurate model such as the one we present in the previous section.

3. EVALUATION

In this section, we evaluate our feedback-based, adaptive energy consumption model. We first describe our experimental setup and then evaluate each design feature. We compare our results to a static model that we describe in [8]. We omit the description here due to space constraints.

3.1 Methodology

Our experimental setup includes multiple Stargate sensor network gateways and H5550 Ipaqs running Linux 2.14.19, an Agilent 54621A oscilloscope and an Agilent E3648A variable power supply. We use the oscilloscope to profile one of the Stargates. We monitor energy consumption in fixed length intervals of 10 million

Computation Benchmark Set		Communication Benchmark Set	
Application	Time (s)	Application	Time (s)
gsmdecode	1.0	game of life (MPI)	9.02
gsmencode	1.1	pvkxb (MPI)	35.2
jpegdecode	5.4	pvnx (MPI)	37.78
jpegencode	17.1	pvkx (MPI)	69.36
mpegdecode	72.9		
mpegencode	91.7		
em3d (Java)	12.1		
bisort (Java)	20.4		
treeadd (Java)	3.8		

Table 1: Benchmarks. We use two benchmark sets. One set (to the left) to model/evaluate computation; another for communication.
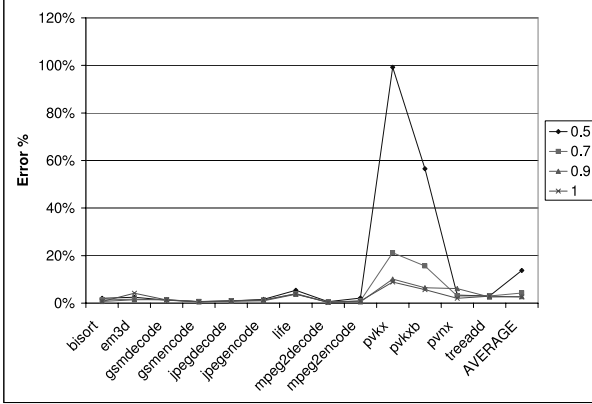

Figure 1: Decay Factor vs. Accuracy. A lower γ gives exponentially greater weight to most recent data. However, this makes the algorithm more vulnerable to noise. Our results show that our algorithm works best when γ is between 0.9 and 1.0.

instructions. A device driver on the Stargate configures the hardware performance counters, (i.e. HPM), to generate an interrupt after each interval. The interrupt handler collects HPM data and forces a logic transition on an output port. We use the Agilent oscilloscope to record these transition times and voltage/current data at a rate of 10000 samples/second. Offline, we analyze this data to extract the length of each interval, peak and average power consumption, and total energy consumption.

To remove the noise in our measurements and analysis, we power the Stargate directly using our Agilent power supply. For the same reason, we substitute battery monitor readings with oscilloscope readings. In Section 3.4, we discuss the impact of battery monitoring unit precision on model accuracy.

To evaluate the communication model, we use a Netgear 802.11b network card on each Stargate; the Ipaqs have their own internal 802.11b cards. We configure all the hosts to the 11Mb/s ad-hoc mode, in direct line of sight of each other.

Again, for comparison purposes, we compare our system to a more complex model that we studied in prior work [8]. The model consists of a computation (E_c) and communication (E_n) subcomponents as does the model we present herein. The model is more complex in that each subcomponent considers additional parameters within both subcomponents:

$$E_c(Joules) = \alpha_0 + \alpha_1 x_1 + \alpha_2 x_2 + ... + \alpha_6 x_6$$

$$E_n(Joules) = \alpha_1 x_1 + B_{tx}\beta_1 + B_{rx}\beta_2 + P_{tx}\beta_3 + P_{rx}\beta_4 + K$$

where x_i's are *core clock cycles, instruction cache misses, instructions not delivered, data stalls, instruction TLB misses*, and *data*
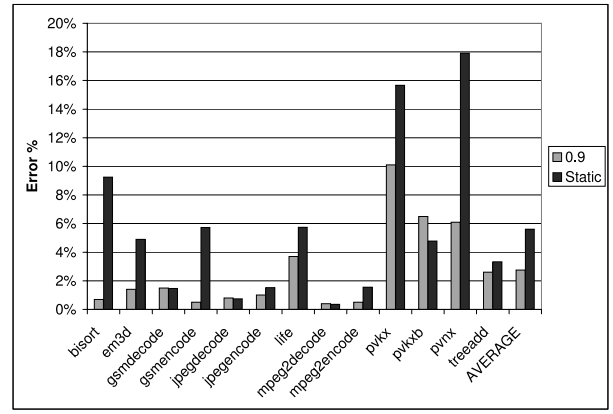

Figure 2: Static vs. Adaptive Models. Figure compares the error rate of the adaptive model with a static one. The adaptive model generates better results in almost every benchmark.

TLB misses, respectively. The α's are computed as described in Section 2.1. In complex communication model, (B_{tx}), and (B_{rx}), specify the transmitted and received bytes as we defined earlier. P_{tx} and P_{rx} are the transmitted and received packet counts, respectively.

To perform our empirical evaluation, we employ the benchmarks in Table 1, that exhibit both computation (left) and communication (right) behaviors. The set includes popular applications from Mediabench [15] as well as Java programs from the JOlden suite [3] that we execute using a Blackdown [1] Java Virtual Machine. The other applications are light-weight distributed programs that use message passing interface (MPI). The MPI applications divide a task into subtasks and distribute them to the other processors. These applications are good candidates for evaluating the energy consumption in programs that perform both computation and communication. We run all our benchmarks on the RAM drive to reduce the effect of flash energy consumption.

Finally, an important challenge inherent in our setup is the monitoring of multiple HPM counters at once. The Intel XScale CPU can monitor only two events at a time, which is less than required by the complex model. As a work around, we execute each benchmark multiple times, collecting one event at a time (i.e. we reserve the other HPM counter for monitoring number of instructions executed). Even though this approach is prone to the effects of runtime variability [11], there is no work-around for this hardware limitation at present. Note that the **compact model** that we present herein does not require repeated runs.

3.2 Decay Factor vs. Accuracy

We first explore the relationship between decay factor, γ, and accuracy. In this experiment, we do not consider execution overhead, hence, we update model coefficients after each interval. We initialize the coefficients differently for computation and communication models. For the former, we use the values described in [8]. For the latter, we monitor a secure file transfer (i.e. scp) of 17MB file across the wireless network multiple times and run the offline profiler to generate the initial values.

Figure 1 shows our results for $\gamma = 0.5, 0.7, 0.9$ and 1. When the decay factor is 1, the algorithm becomes ordinary recursive least squares regression and does not decay any of the previous values. Conversely, when the decay is 0.5, the algorithm effectively remembers only the most recent four measurements. For each benchmark, we show the average estimation error, which we compute using $|measured - estimated|/measured \times 100$. The error rates
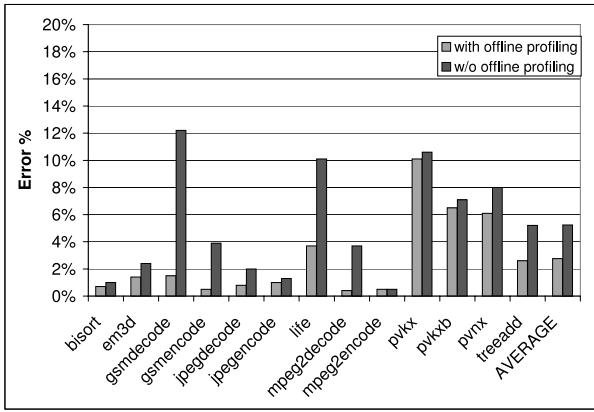
Figure 3: Benefit from an offline profiler. The offline profiler reduces error rate 2.5% in average.

vary from 0.5% to 10% in general, giving an average error of 2.6% for $\gamma \geq 0.9$, and increasing up to 13.7% when $\gamma = 0.5$.

As expected, the error rate is higher for the communication benchmarks. We have encountered a few atypical cases and in one of the benchmarks the error rate was equal to almost 100%, which means the predictions were off by a margin equal to the real value. These atypical cases were specific to low decay factors and to the pvkx benchmark. Pvkx is an MPI program that includes a lot of short communication and computation phases. These phases generate sudden, transient changes in program behavior. When the decay factor is very low, RLS-ED remembers a very short history and reacts much faster than necessary, generating erroneous estimations. In other cases, the error rates are much lower. Overall, moderate decay provides the best result.

In Figure 2, we compare the adaptive model, $\gamma = 0.9$, to a static model from [8]. The dynamic model provides much lower estimation estimation errors in general (2.6% to 5.6%). The only benchmark for which the dynamic model generates higher error is pvkxb (6.5% vs. 4.7%). Pvkxb is similar to the pvkx benchmark, however, it is much shorter. As a result, pvkxb offers very few adjustment opportunities to RLS-ED algorithm.

3.3 Benefits from the Offline Profiler

We next investigate the efficacy of using the offline profiler to reduce model error rate during the initial warm-up period of the RLS-ED algorithm. Figure 3 shows the results across benchmarks for $\gamma = 0.9$. The light colored bars show the error rate when we use an offline profiler, the dark colored bars show the error rate when we do not. For the offline profiler, we determine the coefficients as we outline in Section 3.2. In the absence of the offline profiler, we initialize all the coefficients to 0.

The offline profiler reduces the error rates for all cases. The benefits are more clear for the shorter benchmarks such as gsmdecode, gsmencode, and life. The offline profiler only marginally effects the network benchmarks such as pvkx, pvnx and pvkxb. In contrast to the life application, these three benchmarks tend to transfer smaller amounts of data between their computation period, and are more susceptible to variations in network latency. Overall, profiling reduces error rate from 5.2% to 2.7%.

3.4 Battery Monitor Error Rate vs. Accuracy

A novel feature of our proposed model is the use of the battery monitor as feedback to adjust the model coefficients at runtime. The internal BMU, however, is imprecise, and introduces a much higher error rate than that of our external, high-precision,

equipment. In our target platform, the energy readings are within 4.88mVolts and 0.25mAh (milliampere-Hours) of real voltage and current flow of the battery pack. We assume that the battery voltage stays stable between two readings because of the short period, hence we only consider the current flow measurement errors.

Two other factors, although not directly related to the accuracy of BMU readings, significantly influence our design. The first factor is related to the BMU access overhead. The BMU and CPU is connected through a serial, *one-wire* link and frequent accesses incur an overhead. The second factor is the computational cost of the RLS-ED algorithm. Therefore, we combine ρ *consecutive intervals into a single super-interval* and update the model once for each. In this section, we evaluate several factors of ρ and algorithm accuracy.

The BMU datasheet [7] does not provide any details about measurement error distribution. In our study, we assume a uniform distribution such that the difference between real values and the observed values can be in the range $[-0.125, 0.125]$ mAh. To explore the effect of this error, we injected artificial error into the current flow measurements immediately prior to running the RLS-ED algorithm. We call this amount of error *1X precision.*

To capture future improvements in battery monitoring technology, we also investigate three other precision levels; *2X, 4X, and 8X*. The prefix before *X* is the ratio of reduction in error rate, for example, *2X* has an error range $[-0.062, 0.062]$ mAh. We compare these results to a *precise* battery monitoring unit, which has a precision that is equal to that of our external measurement equipment (no artificial error).

3.4.1 Performance of Complex Model

Table 2 shows our results for three RLS-ED update periods, $\rho = 100, 200,$ and 400. As the unit of ρ is instructions executed, the exact length of update period in wall clock time is somewhat arbitrary. However, in an Intel XScale CPU running at 400 MHz, the updates are separated by at least 10 seconds (much more in practice) when $\rho = 400$, and less for the other cases. In the table, we group the results by ρ and then divide each group into columns of precision levels. The X in the column header shows the precision level. ∞ means that the precision is equal to the external equipment.

We set the decay factor, γ, to 0.9. The offline profiler warmsup the coefficients by running a benchmark once, and then repeatedly runs the benchmark until we monitor at least 2000 intervals. By repeatedly executing them, we can monitor how the feedback and adaptiveness mechanism of the algorithm behaves even for the benchmarks that are shorter than one period.

As the results show, there is a large discrepancy between the *imprecise* (i.e. 1X to 8X) and the *precise* cases. When no measurement errors are present, the RLS-ED algorithm converges quickly, providing estimations that are within 10% of the real values. When measurement errors are present, the estimation error rates increase significantly. This increase is more apparent for some applications such as gsmencode and em3d. These applications have short, sudden changes in their energy consumption behavior. For instance, gsmencode is a very short benchmark with a very smooth execution pattern except the very first few intervals. During these intervals, the energy consumption increase sharply. When these spikes coincide with energy measurements, the RLS-ED detects an immediate increase in energy consumption and overestimates the model parameters.

The effect of ρ on the total system is less obvious, because a higher ρ imposes two different effects. First, as we increase ρ, the relative magnitude of measurement errors asymptomatically decrease. This is a result of the constant error factor that the battery monitor imposes. For instance when precision is $1X$, the expected

	$\rho = 100$ Intervals					$\rho = 200$ Intervals					$\rho = 400$ Intervals				
	1X	2X	4X	8X	∞	1X	2X	4X	8X	∞	1X	2X	4X	8X	∞
bisort	460.2	232.0	61.3	58.6	3.7	1348.9	353.7	273.6	169.2	5.1	31.9	24.1	7.9	6.9	5.4
em3d	1004.4	498.7	184.7	126.2	5.4	709.9	248.0	101.5	91.7	4.3	387.7	148.1	96.8	51.3	3.6
gsmdecode	7.0	4.1	3.4	3.5	3.0	7.0	7.1	6.3	6.0	5.9	12.6	12.2	11.8	11.7	11.6
gsmencode	693.0	295.6	229.3	86.9	0.7	728.6	315.3	183.5	91.3	1.0	1411.3	447.9	419.7	177.7	0.7
jpegdecode	79.9	107.8	31.6	9.9	1.4	49.7	40.6	34.8	6.6	0.7	45.8	13.6	21.2	5.8	0.7
jpegencode	45.8	19.3	14.0	6.1	1.2	25.5	24.0	10.1	4.0	1.4	24.7	10.9	5.1	3.9	1.6
life	149.9	72.3	40.7	19.9	4.3	127.8	48.6	42.6	17.8	3.8	106.0	16.6	8.1	15.1	3.7
mpegdecode	18.2	14.2	5.7	2.4	0.4	22.4	9.6	4.7	2.9	0.4	3.1	0.8	0.7	0.6	0.4
mpegencode	30.8	21.1	11.9	5.0	3.7	34.0	37.1	7.0	5.6	5.6	26.8	17.4	6.2	5.1	4.0
pvkx	69.4	29.9	19.5	13.1	9.4	66.7	19.6	14.7	12.5	8.2	33.5	12.9	10.9	9.9	8.2
pvkxb	49.4	35.1	21.9	17.4	16.1	33.9	28.2	7.9	8.1	6.8	31.3	8.9	8.6	7.2	5.5
pvnx	24.0	29.3	13.2	6.9	5.8	12.1	9.2	6.0	6.0	5.4	6.7	6.4	5.7	5.8	6.0
treeadd	77.8	33.0	18.7	10.0	1.0	91.1	42.6	15.8	12.1	1.4	81.5	43.8	23.4	11.5	1.8
Average	208.4	107.1	50.4	28.1	4.3	250.5	91.0	54.5	33.3	3.8	169.4	58.7	48.2	24.0	4.1

Table 2: Complex model error rates in the presence of an *imprecise* battery monitoring unit. We introduce uniformly distributed noise to the energy consumption measurements as we describe in Section 3.4.

	$\rho = 100$ Intervals					$\rho = 200$ Intervals					$\rho = 400$ Intervals				
	1X	2X	4X	8X	∞	1X	2X	4X	8X	∞	1X	2X	4X	8X	∞
bisort	5.9	2.8	2.3	2.4	1.8	11.3	3.8	2.1	2.3	1.9	7.8	4.6	5.6	3.3	2.9
em3d	165.8	130.1	64.8	114.6	2.7	280.9	58.8	46.3	16.3	2.2	91.9	79.2	8.5	7.8	2.5
gsmdecode	2.5	1.1	0.8	0.7	0.3	2.9	1.4	0.9	0.7	0.5	2.1	1.2	1.2	0.9	0.8
gsmencode	136.3	90.3	17.9	19.3	0.3	298.4	118.6	49.1	20.5	0.4	43.4	58.3	6.6	34.5	0.5
jpegdecode	87.4	23.6	23.0	9.9	1.0	19.6	22.8	9.1	14.2	1.2	18.7	47.6	25.6	12.0	0.9
jpegencode	32.5	10.4	7.4	7.1	4.1	23.5	4.4	3.7	3.3	2.0	20.2	5.7	4.8	6.0	2.9
life	145.3	50.2	19.6	23.3	4.7	90.1	44.5	17.0	10.6	4.8	88.7	7.2	29.2	9.3	4.4
mpegdecode	10.7	2.8	1.9	1.5	0.3	5.0	2.8	1.7	0.9	0.3	1.9	1.0	0.5	0.4	0.3
mpegencode	16.9	10.7	6.1	5.0	2.0	10.0	5.7	4.0	4.2	4.3	11.8	8.4	6.0	5.6	5.4
pvkx	16.6	14.3	13.2	9.7	8.7	40.1	12.2	14.1	9.8	7.2	36.0	24.0	12.2	8.5	7.3
pvkxb	32.7	31.1	19.6	20.6	20.2	51.4	22.0	10.2	7.8	6.6	14.9	18.2	7.7	6.8	5.2
pvnx	13.1	7.2	3.6	2.5	1.1	16.0	5.3	2.7	1.8	1.2	3.7	5.7	2.5	1.5	1.3
treeadd	28.4	14.4	8.7	4.3	2.1	45.6	11.5	7.7	3.3	2.1	88.9	25.2	8.0	4.2	2.2
Average	53.3	29.9	14.5	16.9	3.8	68.8	24.1	12.9	7.3	2.7	33.0	22.0	9.1	7.7	2.8

Table 3: Proposed model error rates in the presence of an *imprecise* battery monitoring unit. We introduce uniformly distributed noise to the energy consumption measurements as we describe in Section 3.4.

error rate is $(0.25)/E$. As a higher ρ means a larger observation period, E becomes larger and error rate becomes smaller. Second, a higher ρ means less frequent model updates, giving the model less chance to react when program behavior changes. Our results show that $\rho = 400$ is better, however, the best ρ varies from one benchmark to another.

3.4.2 Performance of Compact Model

Table 3 shows the same results for the compact model. The compact model error rate shows a significant improvement over the complex model especially when there are feedback errors. For example, when $\rho = 100$ and precision is $1X$ the average error rate is 208% for complex model and 53.3% for compact model. We find that the compact model error rate is less than 3% when there are no feedback errors.

The poor performance of complex model is a result of a phenomenon called *multicollinearity*. In the presence of linear dependence between the variables, the recursive estimates of the RLS-ED algorithm converges slowly and produces inaccurate parameter estimations [18]. The presence of errors in battery monitor readings and a large ρ further complicates the model and reduces the accuracy. The compact model, since it has fewer (and many fewer related) parameters, does not suffer from this phenomenon.

The results indicate that the accuracy of the algorithm is highly dependent on the feedback error rate. At the levels of precision available from BMUs in current devices (i.e. 1X) , feedback errors have a significant adverse effect on algorithm accuracy. However, once the precision levels improve to $8X$, or more, the increase in estimation accuracy improves the quality of energy estimates. When BMU precision level is 8X, the error rate drops to less than 8.0% for $\rho = 200$ and $\rho = 400$.

We also find that the selection of ρ is an important factor in
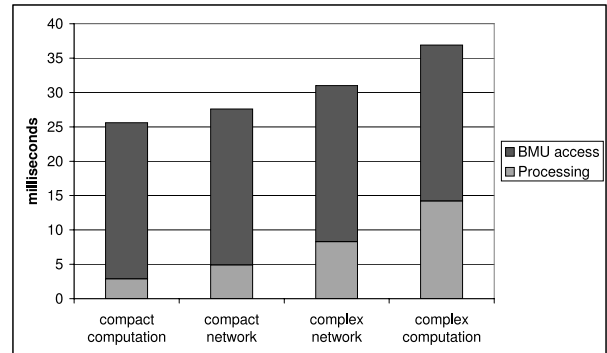

Figure 4: RLS-ED execution cost. The bars show the average CPU time used for each RLS-ED iteration.

model accuracy. Some benchmarks, like em3d and gsmencode, are highly sensitive to the value of ρ. This is a result of short, sudden changes in program behavior (mostly during initialization) that coincide with the RLS-ED updates to the model. Even though we do not evaluate it in the scope of this paper, an application specific selection of ρ may provide better convergence in these cases.

3.5 Execution Cost

Figure 4 shows the cost of executing our model on our target platform. We implemented our model in C as a user-space application. The height of each bar shows the average execution time for a single iteration. The dark colored portion of the bars shows the BMU access time, including the cost of reading data from hardware to kernel space and then transferring to user space. The BMU access time is in average 22.7 milliseconds, and same for all models.

The light colored portion of the bars shows the RLS-ED execution time. For the compact computation model, we measured each iteration to consume 2.9 milliseconds of CPU time. The RLS-ED cost is proportional to the square of model parameter count, and increases up to 14.2 milliseconds for the complex computation model. However, our results show that the dominant cost is the battery monitor access time and not the RLS-ED computation.

4. RELATED WORK

We present a runtime, feedback-based full system energy estimation model for battery powered devices. Our system maps hardware and software counters to power consumption values using first order, linear regression equations. The most closely related work is on HPM-based, static, linear models for CPU and memory energy estimation [4, 12, 17, 13, 10, 6].

In [17], Bellosa et al. reveals the relationship between HPM counters and program energy behavior. [16] improves Bellosa's study by demonstrating that OS system calls can be modeled using their IPC behavior. Their study is simulation based and requires a different model for each operating system routine. [10] estimates energy consumption of major CPU components using their die area, and then monitors their usage rate to estimate CPU energy consumption. Their proposal takes the advantage of sophisticated Pentium-IV HPM mechanism, and not applicable to XScale CPUs. More recently, Bircher et al. [4] demonstrates 2-input linear models for the same Pentium-IV processor.

Unlike the studies above, Contreras et al. [6] focuses on low-power embedded devices. Their work demonstrates a highly accurate (4% error rate) linear regression model for Intel XScale processors. However, this regression model requires monitoring 5 HPM counters simultaneously, which is not possible in XScale processors. To address this, they execute the program multiple times, monitoring one HPM counter at a time. [6] also discusses a HPM-based energy model for memory. However, their memory model is much less precise, due to the lack of memory access counters in XScale HPM design. Prior to [6], a HPM based energy model was demonstrated by [13] on Ultra-SPARC CPU.

One of the primary challenges in our study is to find an optimum set of parameters that explain power consumption, however, simulating each possible parameter combination is hard since there are a large number of factors that shape CPU power behavior. In [14], Lee et al. suggests an efficient and statically sound approach that reduces design space-size considerably. In their simulations, they demonstrate that their regression models can estimate energy consumption with a 4.3% error rate. However, their model does not use any feedback and is evaluated only for a hypothetical CPU. Lee et al.'s approach is complementary to ours as it provides a way for us to design better regression models that uses feedback from battery management unit.

The static models above are undoubtedly useful in characterizing program energy consumption, however they are limited by the static workload that they are developed on. Our study proposes a novel approach to dynamically model and estimate energy consumption on low-power embedded devices.

5. CONCLUSIONS AND FUTURE WORK

This paper presents an adaptive, feedback-based energy estimation model for low-power embedded devices such as HP hand-held computers and Stargate sensor network devices. Our model estimates full system energy consumption of programs using hardware and software counters. Our system starts with an initial model and gradually improves it using dynamic feedback from the battery monitoring unit within the device. We evaluate our model using a large set of applications, and discuss its stability in the presence of measurement errors. Our results show that we can predict energy consumption with 1.0% error rate for computational bound programs and 6.6% error rate for tasks that are both communication and computation bound.

As part of future work, we are extending the power models to include compact flash storage access and LCD display. We also plan to investigate other noise models (such as Gaussian) and other regression techniques that are more tolerant to measurement feedback noise. Finally, we plan to implement our prediction algorithm as a device driver to enable runtime program power optimizations.

6. REFERENCES

[1] Blackdown – http://www.blackdown.org.

[2] M. Anand, E. B. Nightingale, and J. Flinn. Ghosts in the machine: interfaces for better power management. In *Proceedings of the 2nd international conference on Mobile systems, applications, and services (MOBISYS)*, 2004.

[3] B.D.Cahoon. *Effective Compile-Time Analysis for Data Prefetching In java*. PhD thesis, University of Massachusetts at Amherst, 2002.

[4] W. L. Bircher, M. Valluri, J. Law, and L. K. John. Runtime identification of microprocessor energy saving opportunities. In *Proceedings of the International symposium on Low power electronics and design (ISLPED)*, 2005.

[5] Compaq Computer Corporation. *iPAQ Pocket PC*. http://www.compaq.com/products/handhelds/pocketpc/.

[6] G. Contreras and M. Martonosi. Power prediction for intel xscale processors using performance monitoring unit events. In *Proceedings of the International symposium on Low power electronics and design (ISLPED)*, 2005.

[7] Dallas Semiconductors. *DS2760 Data Sheet*. http://pdfserv.maxim-ic.com/arpdf/DS2760.pdf.

[8] S. Gurun and C. Krintz. Full system energy estimation for sensor network gateways. Technical report, UC Santa Barbara, 2006.

[9] Intel Corporation. *Intel Xscale Core*, 2004. Order Number 273473-002.

[10] C. Isci and M. Martonosi. Runtime power monitoring in high-end processors: Methodology and empirical data. In *Proceedings of the 36th ACM/IEEE International Symposium on Microarchitecture (MICRO)*, 2003.

[11] C. Isci and M. Martonosi. Detecting recurrent phase behavior under real-system variability. In *Proceedings of the International Symposium on Workload Characterization (IISWC)*, 2005.

[12] R. Joseph and M. Martonosi. Run-time power estimation in high performance microprocessors. In *Proceedings of the 2001 international symposium on Low power electronics and design (ISLPED)*, 2001.

[13] I. Kadayif, T. Chinoda, M. Kandemir, N. Vijaykirsnan, M. J. Irwin, and A. Sivasubramaniam. vec: virtual energy counters. In *Proceedings of the 2001 ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering (PASTE)*, 2001.

[14] B. Lee and D. Brooks. Accurate and efficient regression modeling for microarchitectural performance and power prediction. In *Proceedings of Architectural Support for Programming Languages and Operating Systems (ASPLOS-XII)*, 2006.

[15] C. Lee, M. Potkonjak, and W. Mangione-Smith. Mediabench: A tool for evaluating and synthesizing multimedia and communication systems. In *Proceedings of the 30th ACM/IEEE International Symposium on Microarchitecture*, 1997.

[16] T. Li and L. John. Run-time modeling and estimation of operating system power consumption. In *ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, 2003.

[17] 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)*, 2002.

[18] P. Young. *Recursive Estimation and Time-Series Analysis*. Springer-Verlag, 1984.