Power Prediction for Intel XScale® Processors Using Performance Monitoring Unit Events

Gilberto Contreras
Electrical Engineering Dept.
Princeton University
Princeton, NJ
gcontrer@princeton.edu

Margaret Martonosi
Electrical Engineering Dept.
Princeton University
Princeton, NJ
mrm@princeton.edu

ABSTRACT

This paper demonstrates a first-order, linear power estimation model that uses performance counters to estimate run-time CPU and memory power consumption of the Intel PXA255 processor. Our model uses a set of *power weights* that map hardware performance counter values to processor and memory power consumption. Power weights are derived offline once per processor voltage and frequency configuration using parameter estimation techniques. They can be applied in a dynamic voltage/frequency scaling environment by setting six descriptive parameters. We have tested our model using a wide selection of benchmarks including SPEC2000, Java CDC and Java CLDC programming environments. The accuracy is quite good; average estimated power consumption is within 4% of the measured average CPU power consumption. We believe such power estimation schemes can serve as a foundation for intelligent, power-aware embedded systems that dynamically adapt to the device's power consumption.

Categories and Subject Descriptors: D.3.4 [**Programming Languages**]: processors — Run-time environments; C.3 [**Special-Purpose and Application-Based Systems**]: Real-time and embedded systems

General Terms: Measurements, Performance

Keywords: XScale, Hardware Performance Counters, Power Estimation, Power Modeling

1. INTRODUCTION

Embedded devices have sprung up in virtually all facets of modern life. With different performance and power requirements than a typical desktop, most portable embedded systems require a very constrained energy profile and efficient use of their energy source.

Efficiency in energy consumption not only comes from the hardware, but also from the software. Software running on embedded devices needs to be designed with power consumption in mind since the processor's power consumption is greatly dependent on its executing workload. Even within a given workload, program performance (and consequently, power consumption) can vary widely, exhibiting "phases" that serve as a signature of the executing workload [1][2][3][4]. Much like performance variations can guide dynamic adaptation [5], knowledge of power behavior at run-time can also be used to change the performance characteristics of software. A fundamental requirement for such adaptation is acquiring fast, low-overhead power consumption estimates of the host system. This paper addresses this requirement.

We present a first-order, linear power estimation model that uses hardware performance counters (HPCs) to estimate run-time CPU and memory power consumption of the Intel PXA255 processor [6]. Our model is designed to take advantage of existing performance monitoring hardware and dominant performance events to allow running applications to estimate their power consumption on-the-fly. Our model links performance an power consumption through a set of *power weights* which are derived only once per processor configuration (voltage and frequency) using offline parameter estimation techniques. We demonstrate how our power estimation model, using a single set of power weights for a given processor configuration, can be used to estimate power consumption of benchmarks that span multiple programming domains including Java CDC, Java CLDC and SPEC2000.

This paper makes the following important contributions:

- Linear parameterization of power consumption based on performance events, with an average estimation error of only 4% across tested benchmarks.
- Parameterization of power at various voltage/frequency settings for the Intel PXA255 processor.
- Lightweight implementation (negligible overhead when sampling counters every 10ms) of our model in C that allows quick estimation of CPU and external memory power at three different core frequencies.

The rest of the paper is organized as follows: We begin in Section 2 by describing related work and highlighting differences between our approach and previous research. Section 3 describes how we link the PXA255's performance metrics with CPU and memory power consumption and the mathematical details of our proposed linear power model. Section 4 reviews our methodology for reading HPCs and obtaining live power measurements. Section 5 describes power estimation results for SPEC2000 and Java benchmarks. Section 6 describes some of the limitations to our approach and how it affects power estimation. We present our conclusions in Section 7.

2. RELATED WORK

Bellosa was one of the first proponents of using hardware performance counters as links to processor power consumption. In [10], Bellosa demonstrates how different performance events such as memory references, floating point operations and L2 cache references exhibit high correlation to processor power consumption. Bellosa first proposed using HPCs to estimate thread power requirements as part of an energy-aware OS scheduler.

Joseph et al. later proposed using HPCs as the basis for entire processor and subcomponent power estimation [11]. This work uses performance counter values as proxies of functional unit usage, which they use in analytical capacitance-based power models to construct a processor power consumption figure. The approach followed by Joseph et al. uses knowledge of circuit-level implementation details (for capacitance models) for increased accuracy, but such technological details might not be readily available for the target processor.

Isci et al. [3] took an alternative approach when estimating the power consumption of a Pentium 4 processor. Isci avoids using circuit-level information by isolating processor functional unit power consumption using a set of specialized micro-benchmarks, access heuristics and live power measurements. Such power decomposition is possible since the Pentium 4 processor has 18 performance counters that can be programmed to monitor up to 59 event classes [12], but it is not applicable for processors like the Intel PXA255 with a small number of available performance events.

Researchers have also investigated ways of estimating power consumption of other parts of the systems using performance counters. Such is the case of [13], where a power estimation methodology for the memory sub-system is examined. In [13] performance counters of the UltraSPARC CPU are used to estimate energy consumption of caches, the memory bus and memory pads. In [14], Li exploits high-correlation between IPC and power consumption to estimate run-time power consumption of OS routines using a power/IPC regression model.

There is no doubt hardware performance counters offer great insight into processor power consumption. While this work shares the notion of leveraging existing processor hardware for power estimation, there are three fundamental aspects that differentiate our work from previous research:

1. We present a power estimation model for the Intel PXA255 processor, a processor with more stringent power requirements and fewer available performance events compared to mid and high-end processors.

2. We extend previous work by constructing a parameterized linear model that lends itself well to different voltage/frequency core configurations.

3. Rather than using processor-specific technology and layout specifications for power modeling, we use parameter estimation to derive a set of power weights solely based on live power measurements and HPC readings, making our methodology extendable for any type of processor with HPCs.

3. POWER ESTIMATION USING HPCS

3.1 CPU Performance Event Selection

The 15 performance events monitored by the Intel PXA255 Processor are designed with CPU performance exploration in mind. Consequently, not all of the events are relevant to power estimation. For our work we have selected performance events with high correlation to power consumption, while avoiding redundancy. Performance events with high correlation to power consumption are more likely to encapsulate power-hungry functional units. Selecting orthogonal events aids in reducing the number of performance events that we need to monitor.

We use five performance events in our linear power model:
Instructions Executed: When comparing performance metrics and power consumption, the metric *Instructions Per Cycle* (IPC) always shows a large positive correlation with power consumption. For example, `REX` — a Java CLDC benchmark — shows a correlation factor of 0.98. This high correlation between IPC and actual power consumption is expected since more functional units are "on" when executing an instruction as opposed when the processor is stalled waiting for some resource to become available.

Data Dependencies: Another performance event with strong (negative) correlation with power consumption is *Data Dependencies.* For example, (`DB` — a CDC Java benchmark — exhibits a correlation factor of -0.85. The number of data dependency cycles provides a reasonable estimate of the total number of cycles that the processor is stalled resolving data dependencies. This is true for relatively small applications since most of the non-IPC cycles will be created by data dependencies.

Instruction Cache Miss: For applications with a large instruction footprint, a significant percentage of stall cycles might be attributed to instruction cache misses. For example, over one quarter of `VORTEX`s cycles go to instruction fetch stalls during the first 50 seconds of execution. The event *data dependencies* does not account for this behavior. In this case, *Instruction cache miss count* can be used to cover fetch unit stall cycles.

TLB Misses: Stall cycles created by instruction and data TLB misses are captured by data stall cycles and fetch stall cycles. TLB misses by themselves, however, have further impact on power consumption since the processor needs to handle memory page table walks. For this reason, we take instruction and data TLB misses into account.

Assuming a linear correlation between performance counter values and power consumption, we can use the following expression for predicting CPU power consumption:

$$\begin{aligned} Power_{cpu} = {} & \alpha_1(IFetch_{miss}) + \alpha_2(DataDep) + \\ & \alpha_3(DataTLB_{miss}) + \alpha_4(InstTLB_{miss}) + \\ & \alpha_5(InstExec) + K_{cpu} \end{aligned} \tag{1}$$

Here, $\alpha_1, \alpha_2, \alpha_3, \alpha_4$ and α_5 are linear parameters or *power weights* and K_{cpu} is a constant representing idle processor power consumption. In reality, some non-linear relationships do exist. This work shows, however, the excellent accuracy of a fully-linear model.

3.2 Main Memory Power Estimation

In addition to CPU power, we also want to track power consumption of external RAM chips for a more comprehensive view of system power requirements.

Detailed power estimation of external memory would require knowledge of SDRAM states and row/column precharge characteristics—events that are not tracked with current CPU performance events. It is possible, however, to estimate SDRAM power consumption by attributing a power cost to every type of SDRAM chip access [9]. It is then a matter of tracking performance events that monitor external memory accesses to estimate SDRAM power consumption. Unfortunately, the Intel XScale microarchitecture revision found in the PXA255 processor does not include any performance event that explicitly accounts for external SDRAM memory accesses. This makes main memory power estimation more difficult than CPU power estimation. There are, however, some performance events that indicate how memory-bound an application is: *Instruction Cache Miss*, *Data Cache Miss* and *Number of Data Dependencies*.

The event *Data Cache Misses* is not a faithful measure of external memory data accesses since memory requests can hit coalescing data buffers that sit between data cache and external memory. The performance event *Instruction Cache miss*, however, generally is a good indicator of memory references due to instruction cache misses. This is explained by noting that the PXA255 processor is a single-issue core, making it necessary to stall until the requested instruction cache line arrives from memory. Since instructions tend to exhibit high spatial locality, this creates a fairly linear relation between instruction cache misses and memory latency (about 120 cycles per instruction cache miss at 400Mhz). Thus, the event *Fetch Unit Stall Cycles* is nearly a linear scaling of *Instruction Cache Miss*.

As mentioned previously, *Data Dependencies* is a performance indicator of the number of cycles the core has to wait for memory dependencies to resolve. Thus, based on the discussion above and with the idea of re-using performance events covered by CPU power estimation, we parameterized memory power consumption using two performance events: *Instruction fetch miss* and *data dependencies*.

$$Power_{mem} = \beta_1(IFetch_{miss}) + \beta_2(DataDep) + K_{mem} \quad (2)$$

3.3 Parameter Estimation

Our next objective is to derive a set of power weights that remain constant under a given CPU performance configuration (frequency and voltage) and that can be used to predict power consumption of a variety of benchmarks and applications running on the PXA255 processor. We achieved this by first deriving power weights for individual benchmarks with the aid of multidimensional parameter estimation. The initial set of weights is then compared with the natural performance characteristics of the benchmark. In other words, power weights that correspond to non-existent or insignificant performance events are zeroed. Once this has been done for a wide selection of benchmarks, corresponding power weights are averaged and adjusted to minimize power estimation errors.

3.3.1 Mathematical Details

A linear estimation of the physical reading η_n based on a linear "weighting" of m parameters $\beta_{n0}, \beta_{n1}, \beta_{n2}, ..., \beta_{nm}$ is given by:

$$\eta_n = X_0\beta_{n0} + X_1\beta_{n1} + X_2\beta_{n1} + ... + X_m\beta_{nm} \quad (3)$$

Which can be written in matrix form:

$$\eta = \mathbf{X}\beta \quad (4)$$

Using linear algebra, one can derive the expression for minimizing the distance between estimation and actual reading. Using an Ordinary Least Squares Estimator (OLS), the expression to minimize is

$$\mathbf{S_{LS}} = (\mathbf{Y} - \mathbf{X}\beta)^{\mathbf{T}}(\mathbf{Y} - \mathbf{X}\beta) \quad (5)$$

where β is a 6×1 column vector containing the 5 unknown power weights plus a constant, $\mathbf{X}$ is a $n \times 6$ matrix comprised of performance readings and $\mathbf{Y}$ is an $n \times 1$ matrix containing the actual power measurements. Minimizing the above equation yields:

$$\mathbf{b_{LS}} = (\mathbf{X^T X})^{-1}\mathbf{X^T Y} \quad (6)$$

Here, $\mathbf{b_{LS}}$ is the set of unique power weights that minimize the least squares error for Equation 3. Matrix $\mathbf{Y}$ is formed by sampling power consumption at 100 samples per second or one sample point every 10ms. To form Matrix $\mathbf{X}$, at least three performance sampling runs are needed because of the limited number of performance counters (2 configurable counters plus one clock counter) available to the PXA255 Processor. Multiple runs may introduce the possibility of time misalignment between samples due to slight sampling variations. This problem was mitigated by introducing a common performance event for each sampling run as described in Section 4.3.

3.4 Parameter Frequency Dependence

Power estimation at different voltage/frequency points is possible by changing the model parameter power weights. This is true since the above procedure will yield a set of power weights that correspond to the particular CPU performance configuration (core frequency and voltage) for which parameters were obtained. Power weights at other performance configurations can then be found by repeating parameter estimation at other processor frequency and voltage settings.

Table 1 shows parameters values for 3 different voltage and frequency configurations of the Intel XScale processor: 200Mhz core at 1.0V, 300Mhz core at 1.1V and 400Mhz core at 1.3V. Parameters displayed in Table 1 assume a PXBus clock frequency and memory clock frequency of 100Mhz. This table can be easily implemented in software or hardware to allow running applications to estimate current processor power consumption in an environment suitable for dynamic voltage and frequency scaling (DVFS).

4. EXPERIMENTAL METHODOLOGY

Our technique uses Equation 1 and 2 to estimate power consumption of the CPU and memory subsystem, respectively. We now describe in detail how performance statistics are collected and live power measurements taken.

4.1 Live Power Measurements

Our hardware testing board consists of the Intel DBPXA 255 Internet Client Development Board [15] running the Linux OS with patched kernel 2.4.19.rmk7-pxa1. The DBPXA255 has two sets of jumpers that facilitate voltage and current measurements of hardware. The first set allows the user to tap into the main power supply of the processor.

Parameter	*Core Voltage/frequency*		
	1.0V 200Mhz	1.1V 300Mhz	1.3V 400Mhz
α_1	1.12×10^{-6}	1.62×10^{-6}	2.30×10^{-6}
α_2	1.11×10^{-8}	1.50×10^{-8}	2.53×10^{-8}
α_3	2.37×10^{-8}	2.42×10^{-7}	1.34×10^{-5}
α_4	4.78×10^{-7}	8.98×10^{-7}	2.11×10^{-6}
α_5	3.83×10^{-8}	5.38×10^{-8}	8.17×10^{-8}
β_1	2.21×10^{-6}	1.72×10^{-6}	2.14×10^{-6}
β_2	2.19×10^{-8}	1.53×10^{-8}	1.33×10^{-8}
K_{cpu}	0.08	0.115	0.165
K_{mem}	0.055	0.055	0.055

Table 1: Power weight parameters used for CPU and memory power consumption estimation at different voltage/frequency CPU configurations.

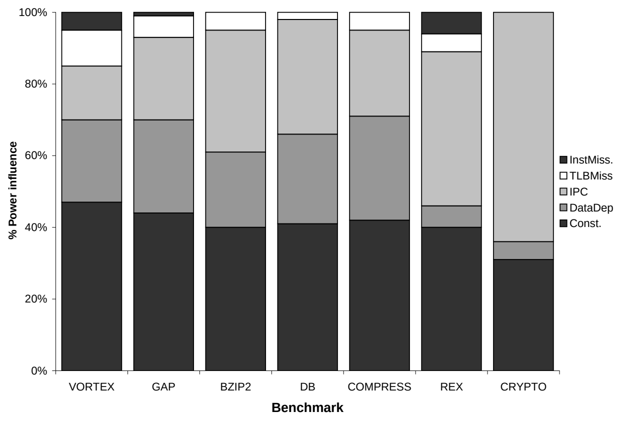

Figure 2: Power breakdown for tested benchmarks.

The second set exposes the CPU's memory voltage supply pins. Thus, these two sets of jumpers allow us to separately measure CPU and memory power.

We used the LeCroy WaveRunner 6030, a quad-channel 2.6G samples/sec oscilloscope for all of our live power measurements. The oscilloscope was triggered into sampling mode by means of a general purpose pin on the processor. A low-to-high logic transition on this general purpose pin initiated sampling of voltage and current consumption. All unused processor units such as the LCD driver, USB unit and the AC97 unit were disabled during live power measurements to more faithfully capture power consumption of the CPU core.

4.2 Accessing HPCs

HPCs are read at the beginning of the main timer interrupt routine of our OS, which executes every 10ms in our testing environment. Sampling code adds only 2% overhead to the OS timer-handler. This overhead is negligible within our sampling window. Counter values are kept in a memory buffer capable of holding up to 6000 samples, which are then saved to a file at the end of the benchmark execution. Since counter values can only be read by privileged instructions, we created an Application Programming Interface (API) comprised of various systems calls which greatly facilitated configuring, reading and writing to the performance monitoring unit from within user space.

To investigate the effects of our added code on power consumption, we performed power measurements of various benchmarks with and without interrupt modifications. We found no significant differences between the two measurements (less than 1% average power consumption difference).

4.3 Run-time Power Estimation

We implemented the linear power models described by Equations 1 and 2, along with our set of final power weights, in a small C program. This C program takes a benchmark application as a parameter and spawns a new thread for the benchmark in order to sample HPCs at regular intervals.

The PXA255 processor can only sample two performance events at any given time. Since Equation 1 requires five performance events for a complete power estimation, multiple runs of the code are needed. For data alignment purposes, however, we run each benchmark four times: On each benchmark run, a pair of performance events is sampled every 10ms, forming a two-column vector of performance readings (E_i, E_j). Each performance pair (E_i, E_j) has a performance event E_c that is common to all pairs. To align a pair of performance vectors (E_1, E_{c1}) and (E_2, E_{c2}), we shift vector (E_2, E_{c2}) until the correlation factor between events E_{c1} and E_{c2} (the common events) is maximum (at least 0.9 correlation factor). At this point, the two vectors are aligned. We selected the performance event *Instructions Executed* as the common performance event, forming the four performance pairs (E_{Imiss}, E_{InstEx}), $(E_{DataDep}, E_{InstEx})$, (E_{DTLB}, E_{InstEx}) and (E_{ITLB}, E_{InstEx}).

5. ESTIMATION RESULTS

5.1 CPU Power Prediction

We tested our linear power estimation model to the test using SPEC2000 (`VORTEX`, `GAP` and `BZIP2`), Java CDC (`DB` and `COMPRESS`) [7] and Java CLDC (`REX` [17] and `CRYPTO` [17]) benchmark. We selected C, Java CLDC and CDC since they represent the common programming domains found in modern embedded systems [8]. For each tested benchmark, we took 50 seconds worth of performance points (5000 samples) or until completion of the benchmark, whichever occurred first.

Figure 1 compares measured and estimated power consumption for `DB`, `VORTEX` and `REX`. From these graphs it is easy to see that our proposed HPC-based power estimation model is able to track CPU power consumption fairly well across time. Average error for these benchmarks is 2%, 1% and 6.5%, respectively. Benchmarks that better expose their characteristics through HPCs tend to have lower estimation errors. We tested our model using other benchmarks from MiBench [16] with very similar results. Average error for all our tested benchmarks is within 4%. Table 2 summarizes the results of our CPU power estimation approach.

The nature of our power model allows decomposition of power consumption among the various measured performance events plus a constant idle CPU power consumption. This is done in Figure 2, where we have decomposed average estimated power consumption of our benchmarks into 5 power contributors. Idle power consumption is responsible for about 40% of the total average estimated power across our tested set of benchmarks. *Data Dependencies* contribute a signif-

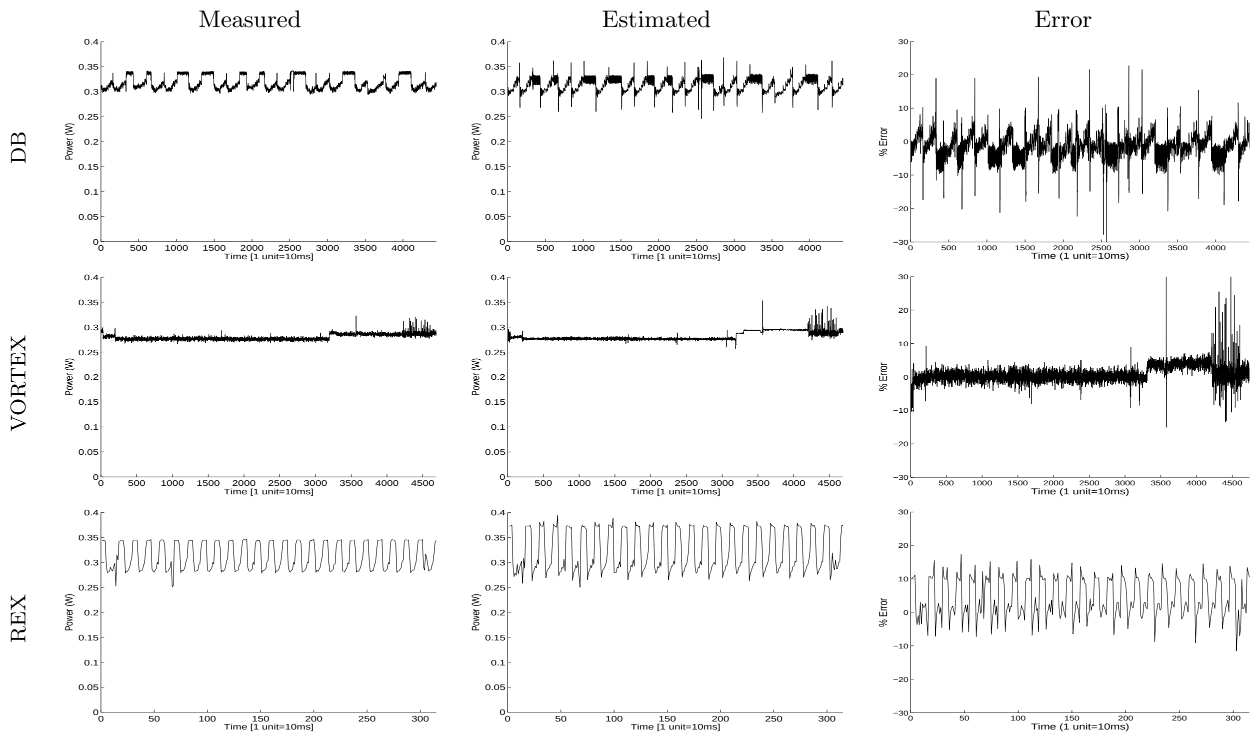

Figure 1: Comparison between measured (left) and estimated (center) power consumption for three benchmarks: DB, VORTEX and REX. Slight misalignment between performance counter readings causes power estimation spikes as seen in DB. The rightmost column shows percent error between estimated and measured power consumption.

icant percentage of power consumption for SPEC2000 and Java CDC benchmarks, which is not the case for Java CLDC applications. Java CLDC applications are small enough for their data set to fit in the data caches, which explains the small quantity of data dependency occurrences. The next big event contributor is *Instructions Executed*, responsible for an average of 24% of the power for SPEC2000 benchmarks, 28% for Java CDC and 54% for Java CLDC. *TLB miss* power consumption occurs in almost all of our tested benchmarks. Our measurements show that SPEC2000 and Java CDC benchmarks suffer mostly from data TLB misses, which account for about 5% of the average power consumptions for this set of benchmarks. `REX`, on the other hand, has no power consumption due to data TLB misses, but instruction TLB misses account for 5% of the total average estimated power. Power usage due to instruction fetch misses seem to be visible only for `VORTEX` and `REX`, with a 5% and 6% power contribution, respectively.

5.2 Memory Power Prediction

Power estimation for external memory was done following the same parameter estimation procedure outlined in Section 3.3. We computed power parameters α_1 and α_2 from Equation 2 under the same performance points described therein. Parameter values are shown in Table 1. Bottom portion of Table 2 shows our memory power estimation results.

Power estimation for external memory using performance events is not as precise as power estimation for the CPU (`CRYPTO` reports up to 70% average error). The foremost reason for this discrepancy is the lack of performance events that track memory transactions executed by the memory management unit. Unlike CPU power estimation parameters, SDRAM power weights increase in magnitude at lower CPU performance points. This is because data dependencies are "more costly" at lower performance points. That is, the processor core stalls for more cycles at 400Mhz than at 200Mhz given approximately the same SDRAM power consumption and same time interval. Power consumption of the SDRAM chips does not vary as significantly as CPU power consumption between different performance points. Since the SDRAM voltage and frequency are maintained at 3.3V and 100Mhz respectively, power variations within performance points are primarily caused by variations in memory access rates.

6. DISCUSSION

This paper has shown how performance counters can be used to recreate faithful depiction of CPU and memory power consumption. There are, nevertheless, steps one could take to further improve this approach.

Dynamic power consumption is one of the major power contributors of the PXA255 processor. This paper has introduced a linear power consumption model that depends on the number of times certain performance events have occurred in the processors core, which assumes, via power weights, a constant activity factor for covered functional units. Variations in dynamic power consumption can potentially be created by sharp variations in data activity factors that are characteristic of certain data sets or instruction se-

Benchmark	*200Mhz, 1.0V*				*300Mhz, 1.1V*				*400Mhz, 1.3V*			
	Measured		Estimated		Measured		Estimated		Measured		Estimated	
	Avg	StdDev	Avg	StdDev	Avg	StdDev	Avg	StdDev	Avg	StdDev	Avg	StdDev
DB	0.118	0.004	0.118	0.004	0.194	0.007	0.193	0.008	0.317	0.013	0.313	0.014
COMPRESS	0.114	0.006	0.116	0.006	0.184	0.012	0.188	0.013	0.309	0.018	0.307	0.020
GAP	0.109	0.006	0.113	0.005	0.168	0.010	0.174	0.009	0.303	0.018	0.291	0.018
VORTEX	0.109	0.002	0.111	0.002	0.171	0.004	0.173	0.005	0.280	0.006	0.282	0.008
BZIP2	0.117	0.012	0.126	0.018	0.189	0.023	0.193	0.036	0.313	0.041	0.287	0.072
REX	0.119	0.008	0.121	0.008	0.190	0.015	0.200	0.023	0.312	0.028	0.320	0.041
CRYPTO	0.122	0.004	0.138	0.007	0.212	0.007	0.242	0.018	0.357	0.014	0.396	0.033
DB	0.074	0.008	0.076	0.006	0.071	0.008	0.073	0.007	0.087	0.008	0.087	0.007
COMPRESS	0.082	0.009	0.081	0.008	0.088	0.010	0.084	.009	0.093	0.009	0.091	0.010
GAP	0.074	0.011	0.078	0.006	0.072	0.011	0.080	0.007	0.076	0.011	0.086	0.007
VORTEX	0.063	0.010	0.060	0.008	0.093	0.010	0.084	0.004	0.086	0.007	0.092	0.004
BZIP2	0.075	0.018	0.075	0.008	0.080	0.018	0.078	0.009	0.077	0.018	0.083	0.011
REX	0.058	0.019	0.070	0.009	0.058	0.022	0.071	0.009	0.063	0.023	0.077	0.012
CRYPTO	0.044	0.010	0.063	0.005	0.037	.009	0.063	0.004	0.038	0.010	0.065	0.005

Table 2: Comparison between measured average and estimated average power consumption for CPU (top) and memory (bottom) for three different frequency settings of the Intel PXA255 processor. Units are watts.

quences. The PXA255 processor's power consumption has been shown to be heavily dependent on activity factors when extreme data sets are used to force worse-case activity factors within the microprocessor [17]. Worse-case activity factors, however, rarely occur in real-life applications.

The Intel PXA255 Processor is a single-issue machine. Its relatively simple architecture combined with aggressive clock gating allows one unit's power consumption to be quite independent from other components. Furthermore, since performance events and functional unit usage have a one-to-one relation, tracking performance events is an indirect way of measuring functional unit usage. This is true when we know the pipeline structure of the processor and the type of instruction that is being executed. The pipeline organization of the PXA255 processor is readily available. However, the type of instruction being executed at any given cycle is much harder to know during runtime, particularly since the PXA255 processor has no performance event that helps us distinguish between an "ADD" and "MUL" instruction, for example. This inability to distinguish the precise instruction being executed as well as its causing effects (cache miss, branch miss, etc) makes it difficult to decompose power consumption to fine-grained architectural functional units. Our approach allows power estimation by assigning "power cost" to the various available performance events rather than to the various microarchitectural functional units. This approach thus places power weights on groups of functional units rather than individual ones, making power estimation on the PXA255 possible.

In spite of the limitations mentioned above, our power prediction model provides a reasonably good estimate of the real CPU power consumption as they capture common performance cases. We feel this model can be used by software designers to get a quick power estimate of applications without the need of special hardware for power measurements.

7. CONCLUSIONS

We have shown how existing hardware performance counters in the PXA255 processor can be used to estimate CPU and memory power consumption in embedded systems. Furthermore, we exploit the Intel PXA255 processor's support for different core voltage/frequency configuration to construct a parameterized power model that allows us to estimate power consumption at various performance points. We demonstrated how five performance events can be used to estimate power consumption within 4% of the average measured power consumption. Implementation of our parameterized power models on the Linux OS provided us with a lightweight power estimation infrastructure, which can be used to obtain quick and accurate power consumption estimates.

8. REFERENCES

[1] E. Duesterwald, C. Cascaval, S. Dwarkadas, Characterizing and Predicting Program Behavior and its Variability *Proceedings of the 12th International Conference on Parallel Architectures and Compilation Techniques (PACT'03)* October 2003

[2] P. Nagpurkar and C. Krintz, Visualization and Analysis of Phased Behavior in Java Programs. *ACM International Conference on the Principles and Practice of Programming in Java (PPPJ)* June 2004

[3] C. Isci and M. Martonosi, Runtime Power Monitoring using High-End Processors: Methodology and Empirical Data, 2003. MICRO'36.

[4] P. F. Sweeney et al., Using Hardware Performance Monitors to Understand the Behavior of Java Applications. *USENIX 3rd Virtual Machine Research and Technology Symposium (VM'04)* May, 2004

[5] A. S. Dhodapkar and J. E. Smith, Managing Multi-Configuration Hardware via Dynamic Working Set Analysis *Proceedings of the 29th annual International Symposium on Computer Architecture (ISCA'02)* May 2002

[6] Intel XScale Microarchitecture for the PXA255 Processor: User's Manual *Intel Corporation*, March 2003. Order No. 278796.

[7] SPEC JVM98 Benchmarks, Standard Performance Evaluation Corporation. http://www.spec.org/osg/jvm98/.

[8] Ulrik Pagh Schultz et al. Compiling Java for Low-end Embedded Systems. *Language, Compiler and Tool Support for Embedded Systems (LCTES03)* June 2003

[9] W. Shiue and C. Chakrabarti, Memory Exploration for Low Power, Embedded Systems. *Proceedings of the 36th ACM/IEEE conference on Design automation* 1999

[10] F. Bellosa, The Benefits of Event-Driven Energy Accounting in Power-Sensitive Systems *Proceedings of the 9th workshop on ACM SIGOPS European workshop* 2002.

[11] R. Joseph and M. Martonosi, Run-time Power Estimation in High Performance Microprocessors *Proceedings of the 2001 international symposium on Low power electronics and Design (ISLPED'01)* 2001.

[12] Intel Corp, Intel Pentium 4 and Intel Xeon Processor Opt. Ref. Man., 2002. developer.intel.com/design/Pentium4/manuals/248966.htm.

[13] I. Kadayif, T Chinoda, M. Kandemir, N. Vijaykrishnan, M. J. Irwin and A. Sivasubramaniam, vEC: Virtual Energy Counters. *Proceedings of the 2001 ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering.* 2001.

[14] T. Li and L. Kurian John, Run-time Modeling and Estimation of Operating System Power Consumption. *ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems* 2003

[15] Intel DBPXA255 Development Platform for the Intel Personal Internet Client Architecture, *Intel Corporation*, February 2003. Order No. 278701-001.

[16] M. R. Guthaus et al. MiBench: A free, Commercially Representative Embedded Benchmark Suite. July 2001. IEEE 4th Annual Workshop on Workload Characterization.

[17] G. Contreras, M. Martonosi, J. Peng, R. Ju and G. Lueh, XTREM: A Power Simulator for the Intel XScale Core. *The 2004 Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES'04)* June 2004