High-level Software Energy Macro-modeling

T. K. Tan[†], A. Raghunathan[‡], G. Lakshminarayana[‡], N. K. Jha[†]

† Dept. of Electrical Eng., Princeton University, NJ 08544
‡ NEC, C&C Research Labs, Princeton, NJ 08540

Abstract

This paper presents an efficient and accurate high-level software energy estimation methodology using the concept of characterization-based macro-modeling. In characterization-based macro-modeling, a function or sub-routine is characterized using an accurate lower-level energy model of the target processor, to construct a macro-model that relates the energy consumed in the function under consideration to various parameters that can be easily observed or calculated from a high-level programming language description. The constructed macro-models eliminate the need for significantly slower instruction-level interpretation or hardware simulation that is required in conventional approaches to software energy estimation.

We present two different approaches to macro-modeling for embedded software that offer distinct efficiency-accuracy characteristics: (i) complexity-based macro-modeling, where the variables that determine the algorithmic complexity of the function under consideration are used as macro-modeling parameters, and (ii) profiling-based macro-modeling, where internal profiling statistics for the functions are used as parameters in the energy macro-models. We have experimentally validated our software energy macro-modeling techniques on a wide range of embedded software routines and two different target processor architectures. Our experiments demonstrate that high-level macro-models constructed using the proposed techniques are able to estimate the energy consumption to within 95% accuracy on the average, while commanding speedups of one to five orders-of-magnitude over current instruction-level and architectural energy estimation techniques.

1. Introduction

In systems or sub-systems implemented as software running on dedicated processors, power consumption depends significantly on the software being executed. Therefore, a paradigm shift towards power estimation from the software standpoint is natural. While power dissipation is a phenomenon that physically occurs in the underlying system hardware, software power (or energy) estimation techniques model it as a function of the software being executed, effectively abstracting out many of the hardware details.

Previous work on power analysis of embedded software is largely based on either instruction-level modeling or structural modeling of the underlying hardware architecture. In order to apply embedded software power analysis to large systems, or in an iterative manner for design space exploration (*e.g.*, in the context of system synthesis tasks such as hardware-software partitioning and mapping), it is necessary to develop estimation techniques that demonstrate significantly higher efficiency while maintaining high accuracy or fidelity of estimation. In this paper, we propose an embedded software energy estimation methodology to address the above issues, based on the use of characterization-based macro-modeling.

1.1 Related Work

Power analysis techniques have been proposed for embedded software based on instruction-level characterization [3, 6, 14, 15, 18] and simulation of the processor architecture [4, 11, 16, 19]. Initial work on instruction-level power modeling and analysis was done in [18], where energy consumption of an embedded software program was computed by summing base energy costs for individual instructions, circuit-state overhead costs for consecutive instruction pairs, and additional penalties due to effects such as pipeline stalls and cache misses. The accuracy of this method was improved by accounting for data dependencies including the effects of instruction and data addresses, register IDs, and operand values [6, 15]. Its efficiency was improved by performing measurements on a limited subset of instructions and instruction sequences [3, 15].

A complementary set of approaches to embedded software power analysis is based on the use of cycle-accurate and structure-aware architectural simulators. Such simulators can identify the architectural blocks that are active in each clock cycle during a program's execution, as well as record the stream of input operands seen by each architectural block. Early work using this approach characterized the power consumption of each architectural block as a single number [11], while subsequent work used macro-modeling concepts from RT-level power estimation to model the power consumption of architectural blocks as functions of their input data values [19]. Structure-aware architectural simulation approaches have been shown to be applicable to modern processor architectures (with deep pipelines, superscalar and out-of-order execution, and branch prediction and speculative execution) [4], and to VLIW processors [16].

In order to estimate and minimize the energy consumption of embedded systems, it is necessary to combine energy estimation tools for the embedded processors sub-system with models and tools for other system components. Tools in

Acknowledgments: This work was supported in part by Army CECOM and in part by DAPRA under contract no. DAAB07-00-C-L516.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
DAC 2001, June 18-22, 2001, Las Vegas, Nevada, USA.
Copyright 2001 ACM 1-58113-297-2/01/0006 ...$5.00.

[2, 7, 8, 17] can be applied to various system-level design tasks, including hardware-software partitioning, configuring parameterizable cores, design of system memory hierarchies, improving battery usage efficiency, operating system power reduction, power-conscious source coding styles, *etc.*

Recent research has recognized the need for efficient power estimation techniques for embedded software [8, 13]. In [8], techniques such as caching (re-using the result of a previous invocation of the instruction set simulator for the same program segment) were used to accelerate system-level power estimation. However, the inherent speed of instruction-set simulation is still a bottleneck. A "power data bank" approach to software energy estimation was proposed in [13]. It exploits the fact that many embedded software programs are constructed using significant use of pre-defined library packages, and some "glue code". The power data bank for a processor contains energy consumption and execution time values for basic instructions, as well as library functions. While it is mentioned in [13] that different instances of a function may have different energy consumption (*e.g.*, due to cache and pipeline effects), their solution is to store a single statistic such as the average observed over all instances of a function, in the power data bank.

1.2 Paper Overview and Contributions

In this paper, we present two different approaches to energy macro-modeling for embedded software that offer distinct efficiency-accuracy characteristics.

- For functions whose computational or algorithmic complexity can be easily expressed in terms of some parameters (*e.g.,* data-intensive functions), we propose complexity-based macro-modeling, where the variables that determine the algorithmic complexity of the function under consideration are used as the macro-modeling parameters. Such energy macro-models require only "black-box" parameters and thus result in highly computationally efficient energy estimation.
- For functions that are not amenable to the complexity-based macro-modeling (*e.g.*, control-intensive functions with highly data-dependent loops and branches), we propose a more general technique, namely, profiling-based macro-modeling where internal profiling statistics for the functions are used as parameters in the energy macro-models. We present several variants of profiling-based macro-modeling, starting from simple basic-block profiling, to different lengths of basic-block correlation profiling and Ball-Larus path correlation profiling.

We have experimentally validated our energy macro-modeling techniques using several embedded software routines on two different processor architectures. They have been evaluated for accuracy and speedup with respect to the low-level power estimator used to construct the macro-models. High-level macro-models constructed using the proposed techniques are able to estimate the energy consumption to within 95% accuracy on the average, while commanding speedups of *one to five orders-of-magnitude* over current instruction-level and architectural energy estimation techniques.

2. General Approach

Our software energy macro-modeling approach uses regression analysis. The idea is to model the software energy consumption using a linear formula:

$$\hat{E} = \sum_{j=1}^{p} c_j P_j \tag{1}$$

where P_j's are the parameters of the macro-model, c_j's are the corresponding coefficients and p is the number of parameters.

The first step in the construction of the macro-model is to determine what parameters are needed to sufficiently model the energy consumption. This step is usually difficult and involves many tradeoffs. We devote Sections 3 and 4 to discuss this problem.

Once the required set of parameters is determined, the next step is to find the corresponding coefficients c_j's. This step can be broken down into a few sub-steps:

1. First, we determine the set of typical input data which is characteristic of the function and the application area of the function. We denote the set of n typical input data for the function as $S = \{I_1, I_2, .., I_n\}$. Usually, n is large enough such that S is representative of the actual application. We then evaluate P_j's for every I_i in S. From this evaluation, we can form a parameter matrix:

$$\mathbf{P} = \begin{pmatrix} P_{1,1} & P_{1,2} & .. & P_{1,p} \\ P_{2,1} & P_{2,2} & .. & P_{2,p} \\ & & \ddots & \\ .. & .. & & .. \\ P_{n,1} & P_{n,2} & .. & P_{n,p} \end{pmatrix} \tag{2}$$

   where $P_{i,j}$ is the j-th parameter value evaluated for input data I_i.

2. We also obtain the energy consumption of the function for every I_i in S, possibly using a low-level software energy estimator. This step is time-consuming. However, this needs to be carried out only once in the process of characterization. We can now form an energy vector: $\mathbf{E} = \begin{pmatrix} E_1 & E_2 & .. & E_n \end{pmatrix}^T$, where E_i is the simulated energy consumption of the function for input data I_i.

3. From Equation. (1), we know that $\mathbf{E}$ and $\mathbf{P}$ are related as follows:

$$\mathbf{E} = \mathbf{PC} \tag{3}$$

   where coefficient vector $\mathbf{C} = \begin{pmatrix} c_1 & c_2 & .. & c_p \end{pmatrix}^T$.

4. $\mathbf{C}$ is obtained using regression analysis [10]:

$$\mathbf{C} = \left[\mathbf{P}^T\mathbf{P}\right]^{-1}\mathbf{P}^T\mathbf{E} \tag{4}$$

Evaluation of the error is important to justify the applicability of the macro-models. We use the following relative error metric:

$$\epsilon = \sqrt{\sum_{i}^{n} \frac{((\hat{E}_i - E_i)/E_i)^2}{n}} \tag{5}$$

where E_i's are the energy data values and $\hat{E}_i$'s are the values given by the macro-model.

Usage of the software energy macro-model to find the energy consumption of a function is essentially a sub-step of the characterization process. Basically, given a particular input data for the function, we evaluate all the parameters P_j's corresponding to it. The estimated energy is then calculated using Equation (1). Suppose the time it takes to obtain the energy estimate using the macro-model is T_{model}. The whole idea of obtaining the energy estimate using the macro-modeling approach is only meaningful if $T_{model} << T_{simulate}$, where $T_{simulate}$ is the time it takes to obtain the energy estimate using a low-level simulator. A key factor to consider is the speedup, where

$$speedup = \frac{T_{simulate}}{T_{model}} \tag{6}$$

3. Complexity-based Energy Macro-modeling

Many frequently used functions in multimedia or other data processing applications make use of some algorithms with known average-case algorithmic complexities. For these types of functions, we can actually base the energy macro-models on these complexities. For example, consider a function that sorts an integer array of size n using the insertion sort algorithm, which has an average-case complexity of $\Theta(n^2)$. We propose the energy macro-model for this function to be:

$$E = c_1 + c_2 s + c_3 s^2 \tag{7}$$

where s is the size of the array. As explained in the previous section, we use regression analysis to obtain the unknown coefficients c_j's. First, we use n different arrays as input data to the function and obtain the corresponding energy consumption values of the function using a low-level software energy simulator. Second, we set up a matrix equation similar to Equation (3) based on the input data:

$$\begin{pmatrix} E_1 \\ E_2 \\ .. \\ E_n \end{pmatrix} = \begin{pmatrix} 1 & s_1 & s_1^2 \\ 1 & s_2 & s_2^2 \\ .. & .. & .. \\ 1 & s_n & s_n^2 \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix} \tag{8}$$

where E_i and s_i are the energy consumption of the function and the array size, respectively, for input array i. The optimal set of c_j's ($j = 1, 2, 3$) can be obtained using regression analysis as in Equation (4). We evaluate the accuracy of the macro-model using the error metric defined in Equation (5).

The advantage of the complexity-based energy macro-model is the ease of use. In the above example, we only need to know the size of the array and the energy consumption of the function can be calculated quickly. In many cases, however, the average-case complexity of the function may not be known. In that case, we need a more general energy macro-modeling technique. Moreover, if the error in the prediction of the macro-model with respect to the modeled data is too large, we know that the average-case algorithmic complexity of the function is not representative of the typical function execution. In this case, we should also resort to a more general macro-modeling technique.

4. Profiling-based Energy Macro-modeling

In this section, we introduce the profiling-based energy macro-modeling technique which is much more general than complexity-based energy macro-modeling.

4.1 Basic-block Profiling

Energy estimation for a software program based on basic-block profiling has been proposed by Tiwari et al. [18]. Basically, the estimated energy is computed as,

$$\hat{E} = e_1 b_1 + e_2 b_2 + ... + Q \tag{9}$$

where b_i's are the basic-block execution counts, e_i's are the instruction energy contribution of the basic blocks, and Q is the extra energy consumed due to cache misses or pipeline stalls. Q is usually not linearly dependent on any of the basic-block execution counts because branch mispredictions and cache misses do not usually occur in a fixed proportionality to the basic-block execution counts. In the prior work, exact cycle-accurate simulation is employed to include Q in the total energy estimate.

The advantage of the regression-based macro-modeling methodology becomes obvious when we want to abstract away all the above details from the high-level energy macro-models. Using basic-block execution counts as the parameters in a linear regression model, the statistical average of the extra energy can be accounted for using basic-block profiling:

$$\hat{E} = c_1 b_1 + c_2 b_2 + ... \tag{10}$$

where b_j's are the basic-block execution counts and c_j's are the regression coefficients to be determined. However, the fitting error can be large (around 10%), depending on the irregularity of the extra energy compared to the instruction energy portion of the basic-block energy.

In some applications, a 10% error in high-level software energy estimation may be acceptable. Therefore, we include the regression-based software energy model based on basic-block profiling in the set of models we propose in this section. The alternative models presented next reduce the errors by partially accounting for the irregularities of the extra energy using a more sophisticated profiling technique called *correlation* profiling.

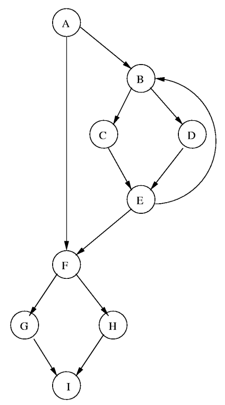
4.2 Correlation Profiling

By correlation we mean a consecutive sequence of events. Correlation-profiling is the counting of such correlations in a program. The concept of correlation has been used in various aspects of performance optimization in microprocessors. For example, Pan et al. [12] used branch correlation to improve the accuracy of dynamic branch prediction. Their results show that, as compared with the traditional two-bit counter-based prediction scheme, the correlation-based branch prediction achieves up to 11% additional accuracy. Mowry et al. [9] used control-flow correlation, self correlation and global correlation to predict data cache misses in non-numeric applications. In particular, the control-flow correlation proposed in [9] is a sequence of basic blocks.

Since correlation profiling has been proven to be effective in predicting branch mispredictions and cache misses, one can expect it to be effective in modeling the extra energy Q as well. Using this idea, we rewrite the energy estimate as

$$\hat{E} = c_1 R_1 + c_2 R_2 + ... \tag{11}$$

where R_j's are the counters for the correlation events and c_j's, again, are the regression coefficients to be determined. Different types of correlation events [9, 12] have been investigated previously in the context of branch and cache miss predictions. In this paper, we concentrate on one particular type, called the control-flow correlation. The basic form of control-flow correlation we use is similar to the form used in [9]. We call it basic-block correlation. We also consider another form of control-flow correlation, which we call *Ball-Larus acyclic path correlation*. We discuss them separately in the next two sub-sections.



BL1: AFGI
BL2: AFHI
BL3: ABCEFGI
BL4: ABCEFHI
BL5: ABDEFGI
BL6: ABDEFHI
BL7: ABCE
BL8: ABDE
BL9: BCE
BL10: BDE
BL11: BCEFGI
BL12: BCEFHI
BL13: BDEFGI
BL14: BDEFHI

(a) CFG (b) All Ball-Larus paths

Figure 1: A CFG with a loop

4.3 Basic-block Correlations

We define an m-block correlation as a sequence of execution of m basic blocks. Profiling of m-block correlations involves counting of these m-block correlation events. For example, profiling of 1-block correlations is actually the same as basic-block profiling. Profiling of 2-block correlations is the same as edge profiling (profiling of edges between basic blocks).

We intend to use fixed-length m-block correlation counts as parameters to be used in the derivation of software energy macro-models. In this context, note that the R_j's in Equation (11) are in fact m-block correlation counts. We use an example to illustrate this idea. Consider a function with the control-flow graph (CFG) shown in Figure 1(a) where A, B, C, ..., I denote basic blocks. A typical set of control-flow traces for this CFG could be:
Trace 1: A F G I
Trace 2: A F H I
Trace 3: A B C E F G I
Trace 4: A B C E B D E F H I
Trace 5: A B D E B D E F G I
⋮
Trace n: ...

Suppose we want to use 3-block correlations as the profiling parameters in our energy model for this function, then we should be counting the occurrence of the following events in the traces: ABC, ABD, AFG, AFH, BCE, BDE, CEB, DEB, EFH EFG, ..., *etc.*

Basically, we enumerate all the 3-block correlations for the particular set of traces. Assuming that there are p different 3-block correlations, then in the same manner as Equation (3), the following matrix equation can be formed:

$$\begin{pmatrix} R_{1,1} & R_{1,2} & .. & R_{1,p} \\ R_{2,1} & R_{2,2} & .. & R_{2,p} \\ & & \ddots & \\ .. & .. & & .. \\ R_{n,1} & R_{n,2} & .. & R_{n,p} \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ .. \\ c_p \end{pmatrix} = \begin{pmatrix} E_1 \\ E_2 \\ .. \\ E_n \end{pmatrix} \qquad (12)$$

where $R_{i,j}$ is the count for the j-th 3-block correlation in trace i, c_j is the model coefficient associated with the j-th correlation, and E_i is the energy consumption of the function associated with trace i. E_i's can be obtained using any low-level software energy estimator [4, 7, 19]. The optimal set of c_j's can be obtained using regression analysis as in Equation (4). Again, we evaluate the accuracy of the macro-model using the error metric defined in Equation (5).

Basic-block correlation profiling can be quite effective in energy estimation for some functions. Since we are only using fixed-length correlations in this method, we expect it to be less effective when the paths through the functions are generally long. To overcome this limitation, we need a profiling method that can capture the notion of path correlations. We discuss this issue in the next sub-section.

4.4 Ball-Larus Path Correlations

Though the term *Ball-Larus path* was coined later, the concept was first described in [1]. Consider the CFG in Figure 1(a) for the purpose of path profiling. The method in [1] considered all possible acyclic paths starting from either the ENTRY (A) node or the nodes which are the targets of one or more back edges, and ending with either the EXIT (I) node or the nodes which are the sources of one or more back edges. These acyclic paths are called Ball-Larus paths. For example, the Ball-Larus paths for the CFG in Figure 1(a) are listed in Figure 1(b).

To go a step further, we also consider the correlations of different Ball-Larus paths. For example, consider two typical traces (the differences are highlighted in bold):
Trace 1: A B C E B C E B **D** E B **C** E F **H** I
Trace 2: A B C E B C E B **C** E B **D** E F **G** I

We can re-write the traces using Ball-Larus paths:
Trace 1: BL7 BL9 BL10 BL12
Trace 2: BL7 BL9 BL9 BL13

In Trace 1, we have two 3-BL-path (3-Ball-Larus-path) correlations, which are BL7-BL9-BL10 and BL9-BL10-BL12. In Trace 2, we have BL7-BL9-BL9 and BL9-BL9-BL13. The energy estimation approach using BL-path correlation profiling is essentially the same as energy estimation using basic-block correlation profiling. Instead of counting the number of basic-block correlations in the traces, we count BL-path correlations.

In the next sub-section, we discuss the considerations we make to determine the "best" profiling method to use.

4.5 Selection of Profiling Methods

Depending on the accuracy and speedup of the macro-models required by the designers, different methods may be chosen. In general, we propose the following procedure:

1. For the function that needs to be characterized, obtain q different energy macro-models using basic-block correlations, 2-block correlations, .., q-block correlations as the profiling methods. Similarly, obtain r different energy macro-models using 1-BL-path correlations, 2-BL-path correlations, .., r-BL-path correlations as the profiling methods. q and r are chosen by the designer who is characterizing the energy macro-model.

2. Pareto-rank the $q + r$ different energy macro-models based on accuracy and speedup. A solution's *Pareto-rank* is the number of other solutions, in the solution pool, which do not dominate it. A solution dominates another solution if it is better than the second one in both accuracy and speedup. For example, if $q = 3$

Table 1: Descriptions of the example functions

Examples	Descriptions	Examples	Descriptions
gcd	calculate the greatest common divisor of two numbers	hash_insert	insert an element into a hash table (linear probing)
hash_search	search for an element in a hash table (linear probing)	igray	compute the gray code of a binary number (N bits)
ins_sort	insertion sort an array of size N	mult	multiply two matrices of size $L \times M$ and $M \times N$
myqsort	quick sort an array of size N	sock_find	find an element in a linked-list
branch	synthetic function with a branch in a loop	br_mem	synthetic function with nested branches in a loop
br_smem	another synthetic function with nested branches in a loop	chksum	calculate the checksum of an integer array of size N
edgedet	edge detection for a gray scale image of size $M \times N$	msort	merge sort an array of size N
myfrag	IP packet fragmentation for a packet of size N		

Table 2: Complexity-based macro-modeling results for SPARClite and SimpleScalar

Examples	Models	SPARClite		SimpleScalar	
		Error	Speedup	Error	Speedup
chksum	$c_1 + c_2N$	1.4%	1361	4.8%	517
igray	$c_1 + c_2log_2(N)$	13.5%	540	16.1%	293
edgedet	$c_1 + c_2M + c_3N + c_4MN$	0.3%	673325	0.6%	313500
ins_sort	$c_1 + c_2N + c_3N^2$	6.9%	30050	6.0%	12473
mult	$c_1 + c_2L + c_3LM + c_4LMN$	2.2%	32213	3.2%	16560
myqsort	$c_1 + c_2N + c_3Nlog_2(N)$	5.6%	38155	3.8%	5419
msort	$c_1 + c_2N + c_3Nlog_2(N)$	4.0%	126780	5.9%	19627
myfrag	$c_1 + c_2N$	4.9%	81517	4.8%	11212

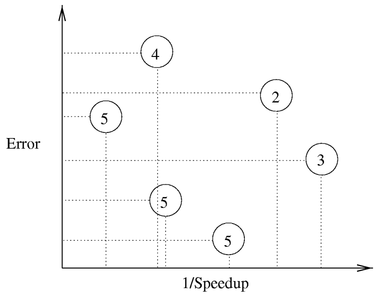


Figure 2: Multiple energy models

and $r = 3$, a typical scenario might be as shown in Figure 2, where the Pareto-ranks of the energy macro-models are indicated in the circles. The set of solutions with the highest Pareto-ranks are the solutions that the designers need to consider. This set is called the Pareto-optimal set in a loose sense.

5. Experimental Results

To demonstrate the feasibility of our macro-modeling methods, we conducted a series of experiments for two types of target processor architectures, Fujitsu SPARClite and SimpleScalar [5]. While SPARClite is a single-issue microprocessor, SimpleScalar is a superscalar microprocessor. By doing experiments on both the platforms, we have attempted to demonstrate the generality of our methods.

For programs executed on Fujitsu SPARClite, we used the instruction-level energy simulator from [7] as our low-level energy estimation framework. For programs executed on SimpleScalar, a few low-level power estimation tools [4, 19] are available. We modified the structure-based low-level power simulator from [4] to make it our low-level software energy simulation framework for SimpleScalar.

In the next two sub-sections, we show the experiments we have done for a few example functions. A description of the functions used in the experiments is given in Table 1.

5.1 Complexity-based Energy Macro-model

We conducted a series of experiments on some software functions to evaluate the feasibility of our complexity-based energy macro-modeling technique. Table 2 shows the macro-model we selected for each of the test functions and the corresponding error. From the error results, we see that the error of these macro-models is actually quite acceptable, considering that we obtain a tremendous speedup over energy estimation using the low-level technique. Of course, in case when the error of such a macro-model is too large, we can conclude that the average-case complexity of the algorithm used in the function is not representative of its typical executions. In such an instance, we should resort to the profiling-based macro-modeling technique.

5.2 Profiling-based Energy Macro-model

We also conducted a series of experiments on some functions to evaluate the feasibility of our profiling-based energy macro-modeling technique. We selected $q = 5$ and $r = 5$. Tables 3 and 4 show the Pareto-optimal set of energy macro-models, and the corresponding errors and speedups. Table 3 shows the results for SPARClite whereas Table 4 shows the results for SimpleScalar. In both the tables, we denote the m-basic-block profiling method by m-BB, and m-Ball-Larus-path profiling method by m-BL. 1-BB stays consistently in the Pareto-optimal sets for all the examples because it is never dominated by the more sophisticated methods, which do not always improve accuracy even after sacrificing in speedup. However, experimental results also show that, in many cases, improvement in accuracy can be obtained using more sophisticated profiling methods.

6. Conclusions

We presented two kinds of macro-modeling techniques for high-level energy estimation for software functions. Both are based on linear regression models. Complexity-based macro-modeling uses the algorithmic complexity of the functions as a hint to the designers to determine the macro-model template. Energy estimation using this approach is very efficient because only easily obtainable information is used in the energy formula. Profiling-based macro-modeling uses either the basic-block correlation or the Ball-Larus path correlation counts as parameters in linear regression models to estimate the energy. We have shown that correlation profiling does lead to an improvement in energy estimation accuracy versus the basic-block profiling method. Our experiments also demonstrated that the speedup of our energy estimation techniques over the low-level technique ranges from one to five orders of magnitude. This speedup can be further improved by optimizing the profiling engine.

Table 3: Profiling-based macro-modeling results for SPARClite

Examples	Method	Error	Speedup	Method	Error	Speedup
gcd	1-BB	0.6%	40.8			
hash_insert	1-BB	0.1%	71.2			
hash_search	1-BB	0.5%	176.7	2-BB	0.2%	169.3
igray	1-BB	0.6%	46.0			
ins_sort	1-BB	7.4%	118.8	3-BB	7.2%	112.8
mult	1-BB	0.4%	175.1			
sock_find	1-BB	3.3%	84.2			
branch	1-BB	4.3%	46.5	3-BB	0.1%	44.2
br_mem	1-BB	6.2%	32.5	5-BL	4.2%	11.6
br_smem	1-BB	3.2%	29.8	4-BB	0.1%	14.7

Table 4: Profiling-based macro-modeling results for SimpleScalar

Examples	Method	Error	Speedup	Method	Error	Speedup
gcd	1-BB	22.1%	14.5	3-BB	11.4%	12.3
	4-BB	8.3%	11.1	5-BB	6.5%	10.2
	3-BL	6.0%	5.8	4-BL	5.1%	5.1
	5-BL	4.6%	4.0			
hash_insert	1-BB	4.6%	20.3	5-BB	3.8%	19.3
	2-BL	3.2%	6.0	5-BL	2.2%	5.9
hash_search	1-BB	2.0%	51.9	3-BB	0.4%	50.2
igray	1-BB	1.3%	24.7	3-BB	1.2%	21.1
ins_sort	1-BB	7.0%	50.0			
mult	1-BB	2.4%	88.4	2-BL	2.3%	27.0
	5-BL	1.6%	26.5			
sock_find	1-BB	3.6%	22.0	5-BB	3.1%	21.9
	5-BL	2.4%	7.0			
branch	1-BB	2.5%	14.4	3-BB	1.7%	13.8
br_mem	1-BB	9.8%	12.0	5-BL	7.5%	4.2
br_smem	1-BB	4.9%	10.6	4-BB	3.2%	5.4

References

[1] T. Ball and J. R. Larus. Efficient path profiling. In *Proc. 29th Int. Symp. Microarchitecture*, pages 46–57, Dec. 1996.

[2] L. Benini and G. De Micheli. System level power optimization: Techniques and tools. *ACM Trans. Design Automation of Electronic Systems*, 5(2):115–192, Apr. 2000.

[3] C. Brandolese, W. Fornaciari, F. Salice, and D. Sciuto. An instruction-level functionality-based energy estimation model for 32-bit microprocessors. In *Proc. Design Automation Conf.*, pages 346–351, June 2000.

[4] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *Proc. Int. Symp. Computer Architecture*, pages 83–94, June 2000.

[5] D. Burger and T. M. Austin. The SimpleScalar tool set, version 2.0. Technical Report 1342, University of Wisconsin-Madison Computer Science Department, June 1997.

[6] N. Chang, K. Kim, and H. G. Lee. Cycle-accurate energy consumption measurement and analysis: Case study of ARM7TDMI. In *Proc. Int. Symp. Low Power Electronics and Design*, pages 185–190, Aug. 2000.

[7] R. P. Dick, G. Lakshminarayana, A. Raghunathan, and N. K. Jha. Power analysis of embedded operating systems. In *Proc. Design Automation Conf.*, pages 312–315, June 2000.

[8] M. Lajolo, A. Raghunathan, S. Dey, and L. Lavagno. Efficient power co-estimation techniques for system-on-chip design. In *Proc. Design & Test Europe*, pages 27–34, Mar. 2000.

[9] T. C. Mowry and C. Luk. Predicting data cache misses in non-numeric applications through correlation profiling. In *Proc. Int. Symp. Microarchitecture*, pages 314–320, Dec. 1997.

[10] R. H. Myers. *Classical and Modern Regression with Application.* Durbury Press, Belmont, CA, 2nd edition, 1989.

[11] P. W. Ong and R. H. Yan. Power-conscious software design - A framework for modeling software on hardware. In *Proc. Int. Symp. Low Power Electronics and Design*, pages 36–37, Oct. 1994.

[12] S. Pan, K. So, and J. T. Rahmeh. Improving the accuracy of dynamic branch prediction using branch correlation. In *Proc. ASPLOS-V*, pages 76–84, Oct. 1992.

[13] G. Qu, N. Kawabe, K. Usami, and M. Potkonjak. Function-level power estimation methodology for microprocessors. In *Proc. Design Automation Conf.*, pages 810–813, June 2000.

[14] J. Russell and M. Jacome. Software power estimation and optimization for high-performance 32-bit embedded processors. In *Proc. Int. Conf. Computer Design*, pages 328–333, Oct. 1998.

[15] A. Sama, M. Balakrishnan, and J. F. M. Theeuwen. Speeding up power estimation of embedded software. In *Proc. Int. Symp. Low Power Electronics and Design*, pages 191–196, Aug. 2000.

[16] M. Sami, D. Sciuto, C. Silvano, and V. Zaccaria. Instruction-level power estimation for embedded VLIW cores. In *Proc. Int. Wkshp. Hardware/Software Codesign*, pages 34–38, Mar. 2000.

[17] T. Simunic, L. Benini, and G. De Micheli. Cycle-accurate simulation of energy consumption in embedded systems. In *Proc. Design Automation Conf.*, pages 867–872, June 1999.

[18] V. Tiwari, S. Malik, and A. Wolfe. Power analysis of embedded software: A first step towards software power minimization. *IEEE Trans. VLSI Systems*, 2(4):437–445, Dec. 1994.

[19] W. Ye, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin. The design and use of SimplePower: A cycle-accurate energy estimation tool. In *Proc. Design Automation Conf.*, pages 340–345, June 2000.