Signature-based Microprocessor Power Modeling for Rapid System-level Design Space Exploration

Peter van Stralen Andy D. Pimentel
Computer Systems Architecture group
Informatics Institute, University of Amsterdam
Email: {pstralen,andy}@science.uva.nl

Abstract

This paper presents a technique for high-level power estimation of microprocessors. The technique, which is based on abstract execution profiles called 'event signatures', operates at a higher level of abstraction than commonly-used instruction-level power simulators and should thus be capable of achieving good evaluation performance. We have compared our power estimation results to those from the instruction-level simulator Wattch. In these experiments, we demonstrate that with a good underlying power model, the signature-based power modeling technique can yield accurate estimations (a mean error of 5.5 percent compared to Wattch in our experiments). At the same time, the power estimations based on our event signature technique are at least an order of magnitude faster than with Wattch.

1 Introduction

The increasing complexity of modern embedded systems, which are more and more based on heterogeneous MultiProcessor-SoC (MP-SoC) architectures, has led to the emergence of system-level design. A key ingredient of system-level design is the notion of high-level modeling and simulation in which the models allow for capturing the behavior of system components and their interactions at a high level of abstraction. As these high-level models minimize the modeling effort and are optimized for execution speed, they can be applied at the early design stages to perform, for example, architectural Design Space Exploration (DSE). Such early DSE is of eminent importance as early design choices heavily influence the success or failure of the final product.

The Sesame modeling and simulation framework [9] provides efficient system-level design space exploration of embedded multimedia systems, allowing rapid performance evaluation of different architecture designs, application to architecture mappings, and hardware/software partitionings. Key to this flexibility is the separation of application and architecture models, together with an explicit mapping step to map an application model onto an architecture model.

Until now, the Sesame modeling and simulation framework has purely focused on the performance analysis of multimedia MP-SoC architectures. Evidently, to make good design trade-offs, also power consumption needs to be taken into account during the process of DSE. Therefore, this paper presents the first step towards including system-level power models in Sesame. More specifically, we introduce the concept of *event signatures* that allows for high-level power modeling of microprocessors (and their local memory hierarchy). As this signature-based power modeling operates at an even higher level of abstraction than commonly-used instruction-level power models, it is well suited for rapid system-level DSE. Using several experiments, we compare the results from our signature-based power modeling with those from Wattch [4], which is a widely-used instruction-level power analysis tool. In order to perform system-level power modeling of an entire MP-SoC, the next step (not addressed in this paper) will be to extend the power modeling framework with models for the interconnect and possible dedicated components in the MP-SoC.

In the next section, we briefly describe the Sesame framework. Section 3 introduces the concept of event signatures and explains how they can be used for high-level power modeling of microprocessors. In Section 4, we describe the power models used for modeling different aspects of microprocessors. Section 5 presents a number of experiments in which we compare the results from our models against those from Wattch. In Section 6, we describe related work, after which Section 7 concludes the paper.

2 The Sesame environment

To facilitate flexible performance analysis of embedded (media) systems architectures, the Sesame modeling and simulation environment [9] uses separate application and architecture models. An application model describes the functional behavior of an application while the architecture model defines architecture resources and captures their per-

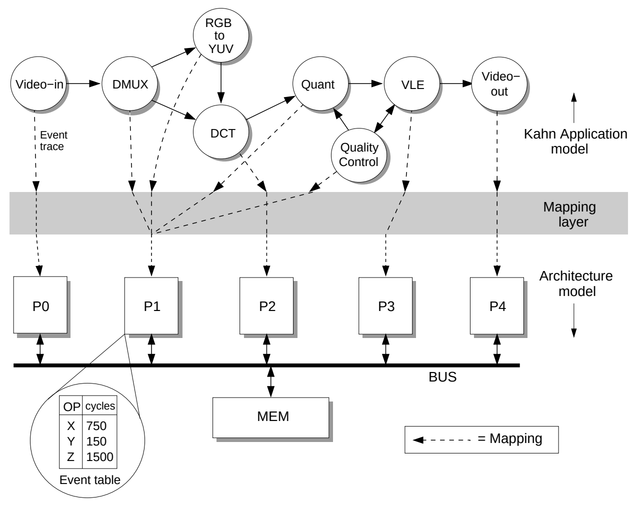


Figure 1. Modeling an Motion-JPEG application on a bus-based MP-SoC architecture in Sesame.

formance constraints. After explicitly mapping an application model onto an architecture model, they are co-simulated via trace-driven simulation. This allows for evaluation of the system performance of a particular application, mapping, and underlying architecture. Essential in this methodology is that an application model is independent from architectural specifics and assumptions on hardware/software partitioning. As a result, a single application model can be used to exercise different hardware/software partitionings and can be mapped onto a range of architecture models, possibly representing different architecture designs or modeling the same architecture design at various levels of abstraction. The layered infrastructure of Sesame is illustrated in Figure 1.

For application modeling, Sesame uses the Kahn Process Network (KPN) model of computation [6], which fits well to the multimedia application domain. In a KPN, parallel processes communicate with each other via unbounded FIFO channels, where reading from these channels is blocking and writing is non-blocking. The computational behavior of an application is captured by instrumenting the code of each Kahn process with annotations that describe the application's computational actions. The reading from and writing to Kahn channels represent the communication behavior of a process within the application model. By executing the Kahn model, each process records its actions in order to generate its own trace of *application events*, which is necessary for driving an architecture model. These application events typically are coarse grained, such as *Execute(DCT)* or *Read(channel_id,pixel-block)*.

An architecture model simulates the performance consequences of the computation and communication events generated by an application model. To this end, each architecture model component is parameterized with an *event table* containing operation latencies (illustrated for Processor 1 in Figure 1). The event table entries could, for example, specify the latency of an *Execute(DCT)* event, or the latency of a memory access in the case of a memory component. The latency values are usually initialized using performance numbers from literature, and can be calibrated using measurements on available hardware or via lower-level simulations of architecture components.

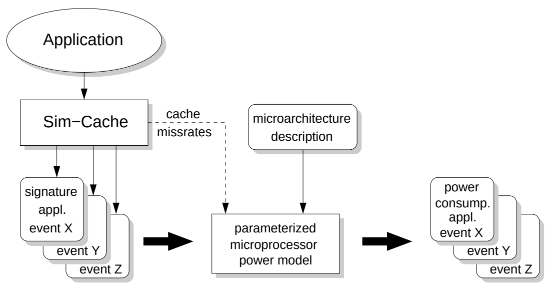
To bind application tasks to resources in the architecture model, Sesame provides an intermediate mapping layer. This layer controls the mapping of Kahn processes (i.e. their event traces) onto architecture model components by dispatching application events to the correct architecture model component. The mapping also includes the mapping of Kahn channels onto communication resources in the architecture model.

Extending the Sesame framework to also support power modeling can be done fairly easily by adding power consumption numbers to the event table. So, this means that a component in the architecture model not only accounts for the timing consequences of an incoming application event, but also accounts for the power that is consumed by the execution of this application event (which is specified in the event table now). The power numbers that need to be stored in the event table can, of course, be retrieved from lower-level power simulators or from (prototype) implementations of components. However, simply adding fixed power numbers to the event table would be a rigid solution in terms of design space exploration: these numbers would only be valid for the specific implementation used for measuring the power numbers. Therefore, we propose so-called *event signatures* to allow for more flexible high-level power estimation in the scope of system-level design space exploration[1].

3 Event signatures

An event signature is an abstract execution profile of an application event that describes the computational complexity of an application event (in the case of computational events) or provides information about the data that is communicated (in the case of communication events). Hence, it can be considered as meta-data to an application event. In this paper, we purely focus on signatures for computational application events (i.e., *Execute()* events). The signatures for these events describe computational complexity in a (micro-)architecture independent fashion using an Abstract Instruction Set (AIS). Currently, our AIS consists of a small set of instruction types such as 'Simple Integer Arithmetic', 'Simple Integer Arithmetic Immediate', 'Integer Multiply', 'Branch', 'Load', and 'Store'. To construct the signatures, the real machine instructions that embody an application event are first mapped onto the AIS, after which

[1]With respect to the rigidness of fixed numbers in the event table, the same reasoning holds for the operation latencies in the table. For this reason, we are also working on performance prediction using event signatures, but this is out of the scope for this paper.



Application
Sim−Cache
cache missrates
microarchitecture description
signature appl. event X
event Y
event Z
parameterized microprocessor power model
power consump. appl. event X
event Y
event Z

Figure 2. Signature-based power modeling.

a compact execution profile is made. This means that the resulting signature is a vector containing the instruction counts of the different AIS instructions.

In Figure 2, signature-based power modeling is illustrated. The Kahn application process for which a power estimation needs to be performed, is simulated using Sim-Cache (which is part of the SimpleScalar suite [1]). Using this (relatively fast) simulator, the event signatures are constructed – by mapping the executed machine instructions onto the AIS as explained above – for every computational application event that can be generated by the Kahn process in question. The event signatures act as input to our parameterized microprocessor power model, which will be described in detail in the next section. For each signature, Sim-Cache may also provide the power model with some additional micro-architectural information, such as cache missrates, branch misprediction rates, etc. In our case, only instruction and data cache missrates are used. The microprocessor power model also uses a micro-architecture description file in which the mapping of AIS instructions onto microprocessor components is described. For example, for the AIS instruction 'Load', it specifies that the ALU is used (to calculate the address), the data cache is accessed, and that the integer register file is read and written. In addition, the micro-architecture description file also contains the parameters for our power model, such as e.g. the dimensions and organization of memory structures (caches, register file, etc,.) in the microprocessor, clock frequency, and so on. Clearly, this micro-architecture description allows for easily extending the AIS and facilitates the modeling of different micro-architecture implementations. The above ingredients (the signatures, additional micro-architectural information per signature, and the micro-architecture description of the processor) allow the power model to produce power consumption estimates for each computational event. Optionally, these power consumption estimates could be stored in Sesame's event tables for re-use purposes.

We note that the generation of event signatures can be performed either statically or dynamically. In static signature generation, Sim-Cache measures the average instruction execution behavior for code fragments that represent application events and constructs the signature based on these averages. So, in this case, the signature generation takes place entirely

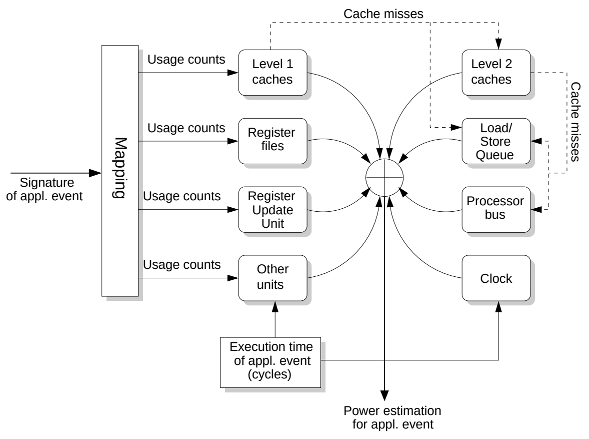


Figure 3. The different components in our microprocessor power model.

off-line. In dynamic signature generation, the signatures are constructed on-the-fly for every application event. This means that the signatures of the same type of application events may change over time due to e.g. data dependent execution behavior inside the code of these events. Another consequence of dynamic signature generation is that Sim-Cache must be co-simulated together with Sesame. The above reasoning also holds for the additional micro-architecture information, like cache missrates, that can be provided by Sim-Cache to the power model. This can also either be done statically (i.e., average based) or dynamically (i.e., exact based).

4 Microprocessor power model

The various components that constitute our microprocessor power model are shown in Figure 3. The power consumption of an application event is calculated by accumulating the power consumptions in each of these components. More specifically, the first step to calculate an application event's power consumption is to map its signature (using the micro-architecture description file, as explained in the previous section) to usage counts of the various processor components. So, here it is determined how often e.g. the ALU (in 'other units' in Figure 3), the register file and the level-1 instruction and data caches are accessed during the execution of an application event. For the memory components (level 1 and 2 caches, register file, etc.), we use Cacti 4.2 [14] to determine the power consumption of read and write accesses to these structures. These power estimates include leakage power. Moreover, we use the cache missrate information provided by Sim-Cache to derive the access counts for the level-2 cache, load/store queue and bus components. Here, we note that we do not (yet) model the main memory since Cacti 4.2 does not support the modeling of DRAMs[2].

[2]As Cacti 5.0 becomes available, the modeling of DRAMs will be possible

The non-memory components in our power model ('other units', 'bus' and 'clock' components in Figure 3) are activity based. That is, they estimate power using the common power equation for CMOS technology:

$$P = \alpha C V^2 f \tag{1}$$

where C specifies the capacitance, V the voltage, and f the frequency. The activity, which is defined as the percentage of transistors which make a switch on each clock cycle, is represented by the parameter α. For the bus component, we use a simple model which abstracts the bus to a set of wires, without any logic, with input and output pins. Currently, we use an I/O pin capacitance of 5 pF per pin and a wire capacitance of 2.15 pF per inch. Further, we assume a wire length of 3 inch and an activity α of 0.5 (half of the wires perform a state switch). For the models of the ALU and multiplier units (in the box 'other units' in Figure 3), we use the capacitance numbers from Wattch [4].

For the power model of the clock component, we base ourselves on the models used in [13, 11]. The model recognizes three sub-components: the clock distribution wiring, the clock buffering and the clocked node capacitance. We assume a H-tree based clock network using a distributed driver scheme (i.e. applying clock buffers). To determine the wiring capacitance, the following equation is used:

$$C_{wiring} = C_{wire} \times \sqrt{A_{die}} \times 2^{N_{tree}-1} \times \sum_{i=1}^{N_{tree}} \frac{1}{2^{\lfloor i/2 \rfloor + 1}} \tag{2}$$

where we retrieve the wire capacitance (C_{wire}) and chip area (A_{die}, determined by accumulating all cache areas) from Cacti 4.2, and calculate the depth of the clock tree (N_{tree}) using:

$$N_{tree} = \sqrt{A_{die} \frac{R_{wire} \times C_{wire}}{Skew_{clock}}} + 1 \tag{3}$$

where resistance R_{wire} is also retrieved from Cacti.

The capacitance consumed by the buffers is modeled to be a fraction of the capacitance consumed by the wiring network. This fraction is dependent on the number of buffers, which is calculated by first taking the ratio of the capacitance of the wiring network and the capacitance of a single buffer. Over this the fourth root is taken, where the value four is actually a parameter, the optimal stage ratio, but this value is fixed within our model.

$$buffers = \sqrt[4]{\frac{C_{wiring}}{C_{single\_buffer}}} \tag{4}$$

$$C_{buffers} = C_{wiring} \times \frac{1}{1 - (\frac{1}{buffers})} \tag{5}$$

For the clocked node capacitance, only memory components are considered. Here, we use the number of read and write ports and the blocksize to calculate the capacitance:

$$C_{clocked} = ports \times blocksize \times C_{trans} \tag{6}$$

The capacity for switching a port is acquired from Cacti, and is equal to the capacitance of a transistor. The clocked node capacitance of each memory structure is summed to the total clocked node capacitance.

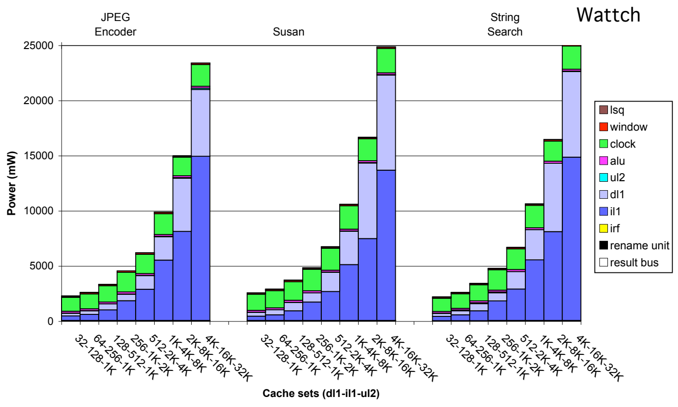
For several components in our power model, the execution time of an application event is needed in order to calculate the activity parameter α. These event execution times may be derived from the event tables in Sesame's architecture model (e.g. in the case signatures are generated statically), or they may be generated dynamically using e.g. our trace calibration co-simulation technique [15].

5 Experiments

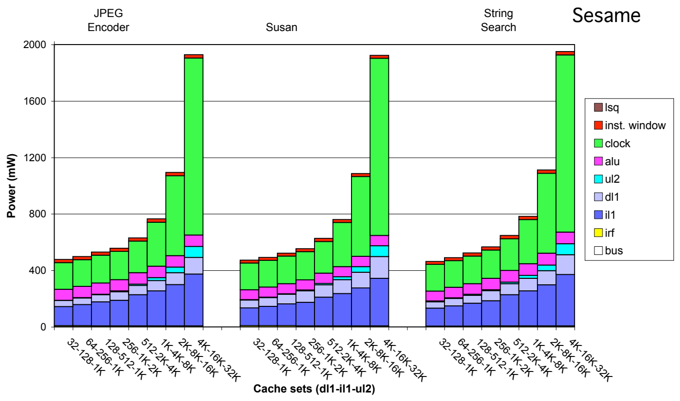
To evaluate our signature-based power estimation, we use three benchmark applications from the MiBench benchmark suite [5]: cjpeg (jpeg compression), susan (edge detection), and string search. We compare our results to those from Wattch [4], which is a widely-used instruction-level power simulator. In these experiments, we use a (in-order issue) PowerPC microprocessor model. Also, we assume a 180nm technology, a voltage of 2.0V, and a frequency of 600MHz.

In the first experiment, we have varied the sizes of the level-1 instruction (*il1*) and data (*dl1*) caches as well as of the unified level-2 cache (*ul2*). This is done by increasing the number of sets in the caches. In Figure 4(a), the power estimation results from Wattch are shown, while Figure 4(b) shows the results from our own power model. Clearly, the absolute power predictions differ significantly between Wattch and Sesame. As will be demonstrated later on, this is mostly due to the differences in the underlying power models. In Wattch, the power consumption is dominated by the level-1 instruction and data caches and, to a lesser extent, the clock network. In Sesame, the power is mostly dominated by the clock network, followed by the level-1 caches. The large discrepancy in absolute power estimations is mainly caused by two differences in the power models of Sesame and Wattch: First, Wattch has a more extensive model of the clocked node capacitance. For example, not only the clocked node capacitance of memory components are modeled, but also the datapath and other components are included. Second, whereas Sesame (in line with Cacti) only models the power consumption related to the critical path inside memory structures, Wattch directly relates the size of a memory structure to the power consumption. The latter can be clearly seen in Figure 4(a) where the power consumption of the level-1 caches rapidly increases when increasing the number of cache sets.

Another observation that can be made is that the power consumption of the clock network in Wattch does not appear to increase for larger cache structures, while there is a significant increase of power consumed by the clock in Sesame. This is due to the fact that Sesame dynamically calculates the



(a)



(b)

Figure 4. Power consumption estimation for Wattch (a) and Sesame (b) when varying the cache sizes.

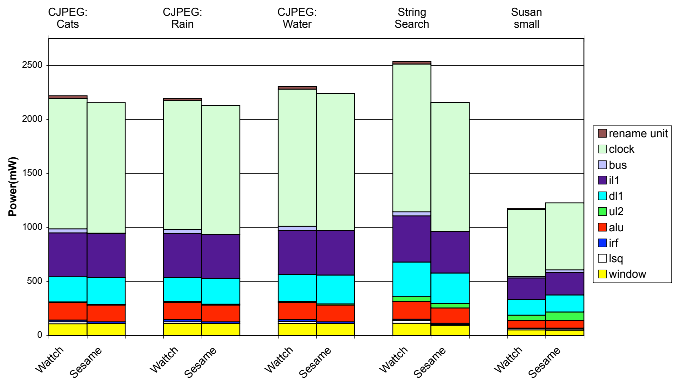


Figure 5. Comparing Sesame and Wattch when the same underlying power models are applied.

die area (using Cacti) for the power modeling of the clock network, while Wattch uses a fixed die area which evidently does not scale with the cache size.

From the above, we learn that although Sesame shows the total power consumption trend reasonably well, its power model still needs to be improved considerably to better reflect the correct absolute power consumptions of the various architectural components. Moreover, the current differences in the underlying power models of Sesame and Wattch make it impossible to actually evaluate the signature-based power modeling technique and the consequences (no notion of separate instructions nor of the data that they use) that come with it. For this reason, we also present an experiment in which we use the access power consumptions from Wattch for the various components in our power model. This way, we try to align both power models such that we can actually measure the effects of the high abstraction level at which Sesame operates. Figure 5 shows the power consumption estimates for both Sesame and Wattch for all three benchmark applications. Here, we have executed cjpeg with three different input pictures of varying complexity (Cats, Rain and Water).

Using these aligned power models, the mean difference between Sesame and Wattch is only 5.5%, with a worst case of 14.9%. Individual components which are data dependent may have a larger mean error, such as 13% for the instruction register file. This is however only a small fraction of the total power consumption. More importantly, for the caches — currently modeled with double-ended bitlines in both Sesame and Wattch, making their access energies largely data independent – the mean error is relative small: 2.1% for the level-1 instruction cache and 6.1% for the level-1 data cache. From these numbers, we can conclude that our high-level method may yield power estimations that are fairly close to those from tools such as Wattch.

Since our initial aim was to speed up the power estimations in comparison to traditional instruction-level power simulators, we also measured the execution times of both the Sesame and Wattch frameworks. Here, we should note that for Sesame we used a set-up in which Sim-Cache was co-simulated together with Sesame to provide our power model with cache missrates (see Section 3). Decoupling Sesame and Sim-Cache would result in even better performance of our power estimations. For the studied benchmark applications, Sesame is on average 10.5 times faster than Wattch. The largest speed-up we measured was 25. Given the fact that – with an appropriate power model – good power estimates can be achieved (see Figure 5), this is a very promising result.

6 Related work

High-level microprocessor power modeling techniques range from analytical methods [3, 8], based on e.g. statistical or information-theoretic techniques, to micro-architecture level instruction set simulators (ISSs) such as Wattch [4], Sim-Panalyzer [2] and SimplePower [16]. Clearly, within this range, there is a trade-off between accuracy and estimation performance. A fair number of efforts also address microprocessor power estimation at a level that is in between analytical and ISS-level models. Most of these efforts estimate power based on a-priori knowledge about instructions or segments of instructions. For example, the power consumption of separate instructions (or instruction pairs [7]) can be measured (using a real processor or a low-level simulator) after which the power consumption of an entire application involves the accumulation of these per-instruction power consumptions [12]. Such measurement-based power estimation can also be performed at a coarser granularity such as at the level of entire functions [10].

In terms of abstraction level, our signature-based power estimation technique is also in between analytical and ISS-based models. But we abstract from single instructions while still applying a micro-architecture level model (with the possibility to perform micro-architectural DSE) to perform power estimation.

7 Conclusions

In this paper, we have presented a technique for high-level power estimation of microprocessors. This technique, which is based on abstract execution profiles called 'event signatures', operates at a higher level of abstraction than commonly-used instruction-level power simulators and should thus be capable of achieving good evaluation performance. The signature-based power modeling technique has been integrated in our Sesame system-level simulation and design space exploration framework and will eventually be extended to allow for system-level power modeling of an entire MP-SoC (i.e., also support the power modeling of the interconnect, shared memory, dedicated IP blocks, etc.).

We compared the results from our signature-based power modeling to those from the instruction-level simulator Wattch. Here, it was shown that although the underlying power model we applied shows approximately the correct overall trends, it needs to be improved considerably to show the correct absolute power consumptions of the various architectural components. However, we also demonstrated that with a good underlying power model, the signature-based power modeling technique can yield accurate estimations (a mean error of 5.5 percent compared to Wattch in our experiments). Important to note here is that the power estimations based on our event signature technique are at least an order of magnitude faster than with Wattch.

References

[1] T. Austin, E. Larson, and D. Ernst. Simplescalar: An infrastructure for computer system modeling. *Computer*, 35(2):59 – 67, Feb. 2002.

[2] T. Austin, T. Mudge, and D. Grunwald. Sim-panalyzer. http://www.eecs.umich.edu/~panalyzer/.

[3] C. Brandolese, W. Fornaciari, F. Salice, and D. Sciuto. An instruction-level functionally-based energy estimation model for 32-bits microprocessors. In *Proc. of the Design Automation Conference*, pages 346–351, 2000.

[4] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *Proc. of the Int. Symposium on Computer Architecture (ISCA)*, June 2000.

[5] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown. Mibench: A free, commercially representative embedded benchmark suite. In *Proc. of the IEEE Workshop on Workload Characterization*, pages 3–14, 2001.

[6] G. Kahn. The semantics of a simple language for parallel programming. In *Proc. of the IFIP Congress 74*, 1974.

[7] M. T.-C. Lee, M. Fujita, V. Tiwari, and S. Malik. Power analysis and minimization techniques for embedded DSP software. *IEEE Trans. Very Large Scale Integr. Syst.*, 5(1):123–135, 1997.

[8] E. Macii, M. Pedram, and F. Somenzi. High-level power modeling, estimation, and optimization. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 17(11):1061–1079, 1998.

[9] A. Pimentel, C. Erbas, and S. Polstra. A systematic approach to exploring embedded system architectures at multiple abstraction levels. *IEEE Trans. on Computers*, 55(2), 2006.

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

[11] N. V. R. Chen and M. Irwin. Clock power issues in system-on-chip designs. In *Proc. of IEEE CS Workshop on VLSI*, pages 48–53, 1999.

[12] J. T. Russel and M. F. Jacome. Software power estimation and optimization for high performance, 32-bit embedded processors. In *Proc. of the Int. Conference on Computer Design (ICCD)*, pages 328–333, Oct. 1998.

[13] N. Sung Kim, T. Austin, T. Mudge, and D. Grunwald. Challenges for architectural level power modeling. In *Power Aware Computing*, pages 48–53. 2001.

[14] D. Tarjan, S. Thoziyoor, and N. P. Jouppi. Cacti 4.0. Technical Report HPL-2006-86, Hewlett-Packard, 2006.

[15] M. Thompson, A. D. Pimentel, S. Polstra, and C. Erbas. A mixed-level co-simulation method for system-level design space exploration. In *Proc. of the IEEE Workshop on Embedded Systems for Real-Time Multimedia (ESTIMedia'06)*, pages 27–32, Oct. 2006.

[16] W. Ye, N. Vijaykrishna, M. Kandemir, and M. J. Irwin. The design and use of simplepower: A cycle-accurate energy estimation tool. In *Proc. of the Design Automation Conference (DAC)*, June 2000.