Run-time Power Estimation in High-Performance Microprocessors

Russ Joseph and Margaret Martonosi
Department of Electrical Engineering
Princeton University
Princeton, New Jersey 08544-5263
{rjoseph, mrm}@ee.princeton.edu

ABSTRACT

Power concerns are becoming increasingly pressing in high-performance processors. Building power-aware and even power-adaptive computer architectures requires being able to track power consumption and attribute energy consumption to the portions of the chip that are responsible for it.

This paper presents the Castle project which aims to deduce the actual runtime power dissipated by different processor units on the CPU chip by leveraging existing hardware. Namely, we examine the use of hardware performance counters as proxies for power meters. We discuss which performance counters count power-relevant events, and how to estimate event counts for power-relevant events not well supported by current, commonly available performance counters. We also discuss sampling-based approaches for estimating signal transition activity within the processor. Overall, we find that these performance counters can be quite useful in providing good power apportionment estimates for programs as they run.

1. INTRODUCTION

Recent efforts in energy-aware computing have examined many different ways to reduce power consumption (for example, [16, 18, 8, 1, 21, 15]). Many of these techniques, particularly ones based on run-time adaptation, require accurate estimates of processor power dissipation. For example, equal-energy operating system scheduling assumes that the operating system can estimate when a processor has used its allotment of energy for this timeslice.

Although power estimates are often crucial for design purposes or for dynamic optimization, they are often quite hard to obtain in a modern, super-scalar processor. Architects typically rely on simulations or on estimates of maximum power dissipated per unit. Compiler writers and operating system developers resort to even higher-level abstractions for power dissipation. While some processors have on-chip thermal sensors, they are often slow to read (requiring an interrupt) and offer little information on which particular unit might be most responsible for power dissipation.

ISLPED'01, August 6-7, 2001, Huntington Beach, California, USA.

Ideally, hardware and software designers want detailed breakdowns of how much power is dissipated, in which units, for individual programs. For example, Figure 1 of [11] presents the power distribution for the Intel Pentium Pro processor, broken down by individual units. That graph was generated by simulation, and in fact such a piechart is extremely difficult to measure directly, since there are no on-chip ammeters to determine how current or power divides between different CPU units.

Our work in developing Castle seeks to provide per-unit power breakdowns based on direct measurements, rather than through simulations or max-power estimates. In particular, this paper outlines and explores a technique for estimating run-time processor power dissipation on a component level. The estimation scheme relies on performance counters, a frequently overlooked facility of most microprocessors. With these counters and register file sampling, we can deduce usage and activity factors for many processor components. The measurement routines are simple, incur only negligible performance and power impact, and—most importantly—are accurate.

The remainder of this paper is structured as follows. Section 2 gives an overview of our approach and outlines key issues in leveraging performance counters for power measurement. Sections 3 and 4 then discuss results for two key types of accuracy: accuracy in unit event counts, and accuracy in signal transition statistics. Section 5 presents the combined effects of examining utilization and signal transitions. Section 6 offers some results on real hardware. In Section 7, we discuss related work, and finally, Section 8 offers our conclusions.

2. POWER MEASUREMENT WITH PERFORMANCE COUNTERS

Our approach allows a run-time monitor or operating system kernel to measure a program's power consumption on the fly. This will create opportunities for interesting energy based scheduling and power optimization techniques. To approximate power consumption, we rely on performance counters to generate usage counts and register values to estimate activity. This measurement scheme does not impact performance because the necessary routines are simple and can be embedded into the operating system's scheduler. Figure 1 outlines our general approach and partial validation scheme.

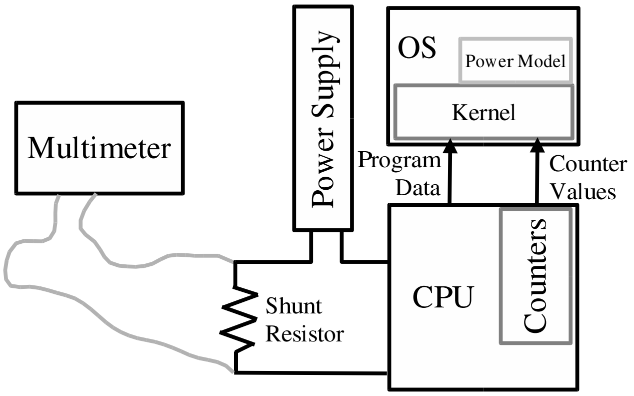

Figure 1: General Approach and Partial Validation.

2.1 Hardware Performance Counters

Most modern microprocessors support programmable event counters that can measure microarchitectural events. Observable events typically include cache misses/hits, bus transactions, branch mispredictions, and instruction retirement. Such event counters are used in commercial performance monitoring toolsets [9] and are also employed in more ad-hoc ways to track subtle performance bugs [12]. The key advantage of hardware performance counters is that instead of relying on simplified performance simulations, programmers can evaluate the impact of their final optimizations on real hardware.

The capabilities provided by event counter mechanisms varies across processor families and within implementations. For example, the Compaq Alpha 21164 features 22 events, but the Alpha 21264 which implements the same ISA, only recognizes nine different events [3]. There are also limitations on how many events may be simultaneously measured. The IBM Power 3-II processor can concurrently measure up to eight of its 238 performance events [3]. The Intel Pentium II processor can only simultaneously observe two out of 77 total events [3].

2.2 Performance-relevant Events vs. Power-relevant Events

Architectural power simulators like Wattch [5] track the utilization of processor components including wakeup logic, functional units, and cache ports. They also monitor switching factors for critical bitlines including register files, result buses, and the store buffer. During simulation, they feed these utilization numbers and activity factors into a high-level power model to estimate the energy behavior of the processor.

In a similar vein, we wish to generate the same kind of power estimates with complete component breakdowns. By sampling performance counters we can approximate the component usage counts. By examining register file contents, we infer bitline switching factors. There are several obstacles that make both tasks interesting. First, the events recorded by performance counters are not the same one might choose for power-relevant counts. This issue is discussed further in Section 3. Second, none of the performance counters are aimed at discerning signal transition activity, which of course is crucial for power measurement. Methods for overcoming this problem are discussed further in Section 4.

Resource	Utilization Formula
rename	insn_decoded
bpred	branch_retired
regfile	(insn_retired - branch_retired) + 0.4 * insn_decoded
icache	fetch_access
dcache	dmem_access
l2cache	l2_access
resultbus	insn_executed
window_preg	5 * insn_executed
window_wakeup	insn_executed
window_selection	insn_decoded
lsq_preg	1/3 * dmem_access
lsq_wakeup	2/3 * dmem_access
ialu	insn_executed - fp_op_exec
falu	fp_op_exec

Table 1: Listing of heuristic approximations for Wattch-Alpha model.

3. POWER-RELATED EVENT COUNTS

3.1 Power Model

In our investigations, we used Wattch [5], an architecture level power simulator. This SimpleScalar[6] based tool leverages analytic models of processor components to generate complete and accurate power statistics. In the spirit of SimpleScalar, Wattch allows a fair amount of flexibility and customization. In most cases, we use Wattch parameterized to resemble an Alpha 21264.

3.2 Which Event Counters to Use?

While most processors have extensive performance counting ability, many of the events that are interesting from a power measurement standpoint do not appear in any performance counter APIs. For example, register file usage is an important contributor to total processor power, but this is not a directly useful statistic from a performance standpoint. As a result, it is not directly available in any performance counter API. To combat this, we use heuristics to infer power relevant counts from the available counters.

Since typical performance counters do not capture all the power relevant events, we must approximate some utilization factors through heuristics. In general, the heuristics depend on machine structure and the available performance counters. We have chosen a set of heuristics that are based on our Wattch-Alpha model and rely on our assumed performance counters. These heuristics may not be suitable on other architectures. For example, Section 5 outlines a different set of heuristics used to model a Pentium Pro. Table 1 summarizes our Wattch-Alpha formulae. We briefly explain the reasoning behind three of our heuristics.

As our first example, we estimate the number of instruction window physical register accesses (window_preg) by multiplying the number of instructions executed by five. This works because the average instruction in our model makes five instruction window/reservation station accesses.

There is no direct way to measure the number of architected register file accesses (regfile). In general, this is difficult to predict since it is partially dependent on early availability of source operands. To first order, it will scale with the number of retired instructions which write values and instructions decoded that read values. The corresponding formula in Table 1 reflects this.

Finally, to estimate the number of wakeup logic accesses (window_wakeup), we substitute with the number of instructions executed. After instructions execute, they alert dependent instructions that their operand values are ready via wakeup logic. So, the number of instructions executed approximates the number of wakeup accesses.

3.3 Counting Limited Events at Once

Unfortunately, the performance counter API only allows us to measure two event types at a time. Even the most accommodating processors only allow eight simultaneous events [3]. To approximate these measurements, we rotate through the available performance counters, examining one pair of event counts at a time. This multiplexing assumes that program behavior is fairly constant with respect to the sampling intervals.

3.4 Validation

Our Wattch simulation model was configured to roughly approximate an Alpha 21264 running at 600 MHz. Unfortunately, the Alpha 21264 has a limited performance counter API, in contrast to its predecessor, the Alpha 21164 which has extensive counter abilities. In response, we have expanded the number of countable events to bring the our Alpha model up to speed. All of the selected performance events can be found in existing processors. Finally, we have assumed aggressive, linear clock gating within the processor.

In our experiments we assume Wattch's basic power model, and rely on the same per usage wattage for most microarchitectural items. This means that we can expect error to be introduced through either sampling or flaws in our heuristic approximations. We assume a 10 millisecond sampling interval. This is an acceptable context switch interval for most operating systems, and we find that it gives good results in practice. We based our analysis on SPEC95 Int and FP benchmarks, compiled by *cc*, with full optimization.

3.5 Results

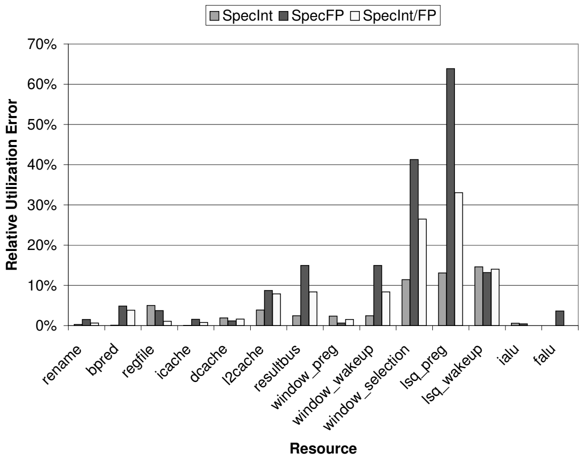

Figure 2: Alpha Model - Resource Utilization Error.

To evaluate our heuristic approximations and sampling approach, we compare them to Wattch's utilization factors on Spec95 benchmarks. Figure 2 shows the relative error introduced by our estimates. Most of the estimated utilization rates match the Wattch measurements. In particular, many of the items including d-cache access and fetch access rates are within 5%. This is to be expected since both of these events are made directly available by the performance counters. Error is only introduced through sampling.

Many of the heuristic based approximations also track very well. In particular, the rename and window physical register rates show little deviation from the Wattch measurements.

Some of the heuristics do not track as favorably. Load/Store Queue (LSQ) physical register accesses are difficult to predict. Our heuristic assumes that these accesses correspond directly to store instructions. Further analysis suggests that this approximation would be fairly robust if we could precisely determine the ratio of load-to-store instructions with the available counters. At the present, limitations force us to chose a fixed ratio, which performs well on most benchmarks, but very poorly on a select few floating point programs.

In addition, window selection is also difficult to predict. This is because selection power is dependent on the number of ready instructions in the instruction window. This is difficult to estimate with the standard counters assumed in this investigation. With additional information on queue occupancy, we could offer a more accurate estimate.

For circuit-based and structural reasons, window selection and LSQ physical register power comprise a relatively small portion of total processor power. As we will show in the following section, these inaccuracies do not severely hamper estimation.

4. SIGNAL TRANSITION STATISTICS

A runtime power estimate based solely on performance counters ignores the variable activity factors dependent on program data. In our studies, we focus on the activity of single-ended bitlines which can comprise a significant portion of total processor power. For example, single-ended bitline power accounts for 11% of the maximum power dissipation in our Wattch-Alpha model. Furthermore, it is impossible to directly measure the crucial activity figures during normal program operation.

Note that the activity on double-ended bitlines is not data dependent since the paired bitlines complement each other. They always have a constant switching factor. As a result, double-ended bitline power can be addressed with performance counter based utilization estimates.

In contrast, the data values seen on the single-ended bitlines determine their power dissipation. If the circuit is precharged to logic one, then only zeros will dissipate power. Conversely, a precharge to logic zero means that only ones will dissipate power. For this type of structure, the population count divided by the bit width yields the activity factor.

To assess activity factors, we sample and average register file population counts when we collect the performance counter values. Then, we scale the maximum power of the single-ended bitline arrays by the estimated activity to produce a total bitline power estimate.

Our activity estimates assume that the register file values are closely coupled to the values passing through the single bitline structures. There is some truth to this since most of the values seen on the physical register file, store buffer, and result bus pass through the architected register file thorough operand reads and writes. In general, we can expect other values to appear on these bitlines, but the majority of data

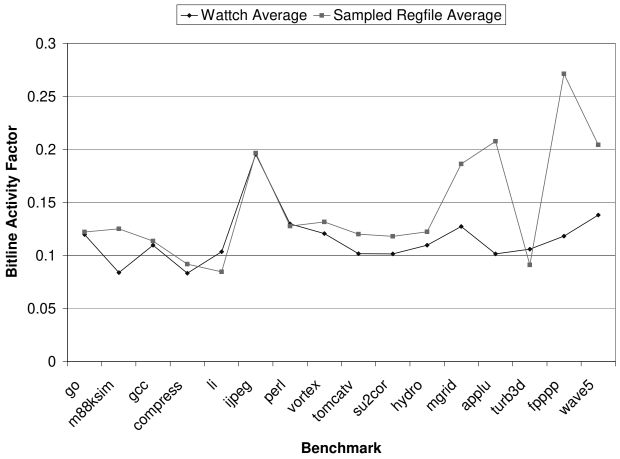

Figure 3: Alpha Model - Estimated Activity.

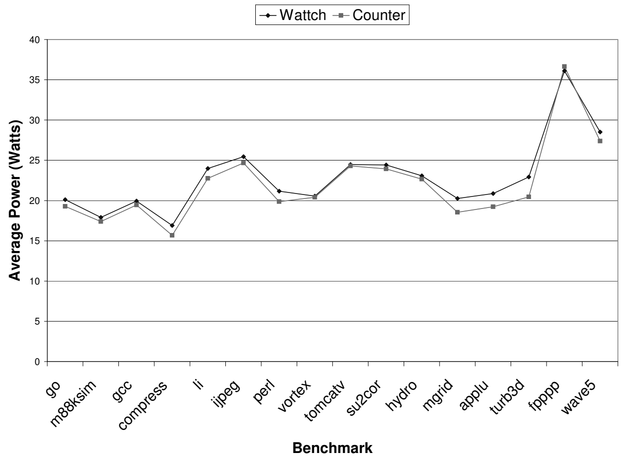

Figure 4: Alpha Model - Overall Estimated Power vs. Measured Power.

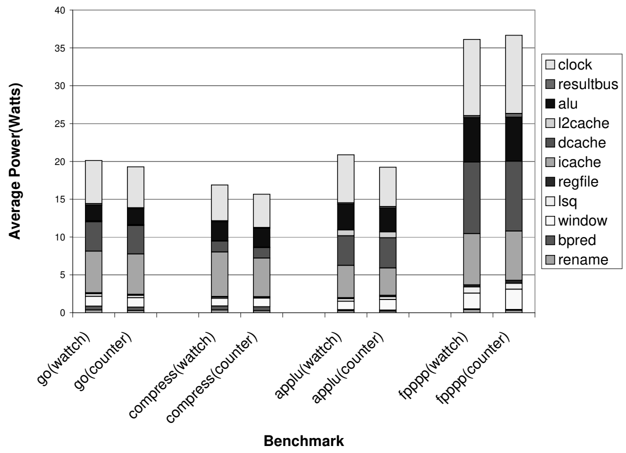

Figure 5: Alpha Model - Component Power Breakdown.

values circulating through the processor will be placed in the register file at some point.

Finally, our scheme assumes that all the registers are used with equal frequency. If a small number of frequently used registers have decidedly different population counts from the rest of register file, our estimate would be skewed. A more sophisticated scheme might use weighted register samples.

4.1 Validation

In our experiments, we used the same Wattch model. On single-ended bitlines, we presuppose that power is linearly dependent on activity factor. The architected and physical register files, store buffer, and result bus are the only single-ended bitline structures in our model. Differences between Wattch's reported power levels and our power estimates will be introduced by way of sampling.

4.2 Results

Within the benchmarks selected for this study, the activity approximation scheme shows varying degrees of success. In Figure 3, the estimated activity factors track the Wattch measured factors on most of the integer benchmarks. On most benchmarks there is less than a 2% relative error. There is a considerable amount of variability on the floating point suite. For fpppp, the approximated activity is as much as 21% greater than the true value. We anticipate activity factors in floating point benchmarks will be more difficult to estimate because they generally use larger bit widths during computation. A significant portion of the integer benchmarks use less than a quarter of the available bit width [4].

5. OVERALL POWER ESTIMATES

Despite small inaccuracies in component usage and activity, the average power estimates depicted in Figure 4 closely follow Wattch's power measurements for all of the benchmarks examined. Fortunately, the largest component usage errors are in units that account for a relatively small amount of the total power. The error introduced by activity is also small for most of the benchmarks. Integer and floating point benchmarks are approximated equally well. Fortunately, discrepancies in floating point activity were not large enough to hamper the estimation scheme in general.

Finally, in Figure 5 we can see that the estimated component breakdowns also match the Wattch results. Error is typically less than 5

6. MEASUREMENTS ON REAL HARDWARE

Obviously, the end goal of our Castle project is to provide real measurements on hardware. Here we present a prototype of our run-time power measurement scheme.

6.1 Methodology

Our platform was an Intel Pentium Pro 200MHz based computer with 128MB and a 2GB hard drive, running Linux 2.2.16. All power measurements were collected by an HP 34401A Digital Multimeter which broadcast readings to a second computer. These samples were then logged and averaged.

To measure total CPU power, we applied an approach similar to [20]. We placed a known shunt resistance between the computer's power supply and the processor's motherboard power terminal as depicted in Figure 1. By observing the voltage over this resistor, we were able to determine the current being drawn and hence the total power consumed. Since the processor and chipset share terminals, we subtracted the chipset power from total power observed and finally deduced the CPU power.

We extended the scheduler inside the Linux kernel to implement our estimation scheme. To sample processor events, we used the Pentium Pro performance counters. We limited our interest to 12 distinct events, and sampled the maximum of two at a time. The events were switched once every timer interval.

Our power estimates were based solely on resource usage. Since IA32 only has eight general purpose integer registers, register based activity estimates seemed inappropriate. A hybrid technique that couples register sampling with pointer analysis and memory data sampling might yield better results.

We used the familiar SPEC Int95 benchmarks in conjunction with the pointer intensive Olden benchmarks[13]. All programs were compiled by egcs-2.91 with the O2 optimization level.

6.2 Modeling Pentium Pro Power

Without a circuit-level knowledge of the Pentium Pro, it was difficult to construct a suitable power model. While we managed to gather maximum component power figures for most of the larger microarchitectural structures including caches, decode logic, and the reorder buffer, we were unable to isolate power estimates for a large number of smaller structures including the BTB and address generation units. In total, these smaller structures constituted as much as 24% of the remaining processor power. With such a large fraction of the microarchitecture unaccounted for, we made the conservative assumption that these unidentified structures had constant, maximum power utilization.

We also assumed that the decoder, issue logic, and reorder buffer were constantly utilized. To calculate memory access and floating point power, we relied on the related performance counter events. For integer functional unit power, we estimated the number of uops decoded and subtracted by the number of executed floating point and memory operations. Finally, we assumed that the global clock distribution power was constant.

6.3 Results

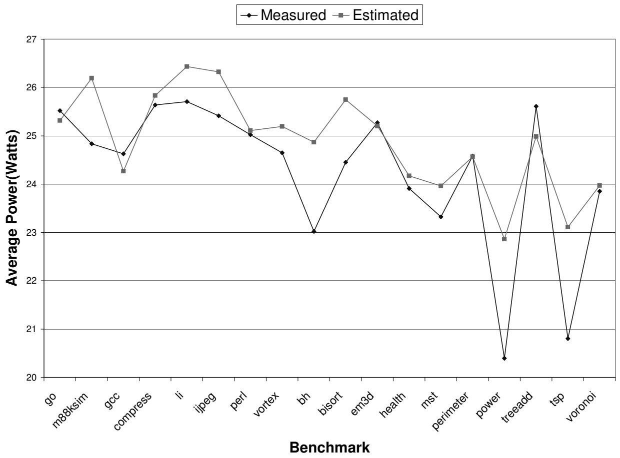

Figure 6: Pentium Pro - Overall Estimated Power vs. Measured Power.

In Figure 6 we plot Castle prototype power estimates versus the direct measurements. Our estimates track the relative power consumption fairly well, considering limited availability of Pentium Pro power data. This prototype could be significantly improved by an enhanced Pentium Pro power model.

One interesting trend is the relatively small differential between the high and low wattage benchmarks in comparison to the Alpha power simulations in Figure 4. The most likely cause is the degree of clock-gating in the two processors. Our Alpha simulation used an aggressively clock-gated processor, but the Pentium Pro in our study makes limited use of conditional clocking.

Our power model makes overestimates for *bh*, *power*, and *tsp*. This is likely caused by the very conservative decoder model. The decode behavior of these three benchmarks differs significantly from the remaining benchmarks. The Pentium Pro decode mechanism is fairly complex, and our initial attempts to reconstruct more robust power models have had limited success. We are currently working on a probabilistic model which may offer better results.

Finally, our Castle based analysis was able to generate component power distributions similar to those in Figure 5. Our future plans include comparing these to a detailed power simulator for the Pentium Pro for more complete validation.

7. RELATED WORK

This research contributes to a growing body of work on microprocessor power measurement and modeling. In pioneering work, Tiwari, et al. explored the instruction level effect of power in microprocessors [16]. This investigation of the Intel 486DX2 identified energy costs for different instruction types, inter-instruction overhead, memory system power, and stall related effects.

This has been followed by many studies exploring instruction level power in embedded microprocessors and DSPs including [10][14]. These efforts produced software power estimate models suited to a particular processor design. These models can be used to predict run-time power and guide energy-aware optimizations [17][18]. Recent work [20][7], has examined dynamic power measurement in pocket computing and personal digital assistants (PDAs). Like many of the earlier studies, this work examined embedded microprocessors and controllers and did not explore the power modeling of wide superscalar machines.

Architectural level power simulators [5][19] have made significant strides toward de-mystifying power dissipation in more complex processors.

In [2], the author presents a run-time power estimation scheme which also uses performance counters. The author observes processor power with a multimeter setup similar to ours. With some empirical data produced by microbenchmarks, energy weightings are devised and assigned to a few critical events. Given these weights and counter sampling, the author outlines an approach for thread-level accounting.

While this proposal offers some novel ideas, it is not complete. Because of the black box approach taken in [2], component power distributions cannot be generated. By leveraging knowledge of the microarchitecture, we can produce more detailed power estimates. In addition, this approach makes no provisions for data-dependent switching activity, which may have a significant impact on program power consumption. By constructing an appropriate power model, and considering data dependent switching factors, we can produce more comprehensive run-time power analysis which may increase the window for power-aware optimization.

8. CONCLUSION

In this paper, we described a general scheme for estimating runtime microprocessor power with performance counters. This approach differs significantly from previous power estimation techniques. It is applicable in high performance processors which exhibit complex energy usage patterns, and more importantly, it provides a rare glimpse of component power consumption. We prototyped our estimation scheme on a Pentium Pro platform.

By leveraging information about machine organization and sizings, one can develop not only a runtime power estimate, but also component power distributions. In addition, register file sampling can approximate bitline activity, and generate even more accurate power estimates. There is however, considerable room for improvement in bitline power approximation as witnessed in the floating point benchmarks in Figure 3. In particular, our current model assumes that all registers are used with equal frequency. By scanning small code snippets, a runtime estimator could generate a more useful weighted average population count. In addition, it might be worthwhile to examine not just the registers, but also frequently accessed portions of memory. A more sophisticated runtime scheme might be able to identify small array based loops, and scan those arrays to produce better population estimates.

Overall, the Castle project offers an interesting hybrid of detailed, but slow power simulations and simple run-time power measurement. Our future plans include both expanding on Castle's accuracy and comprehensiveness as well as applying the approach in power-oriented architecture research.

9. REFERENCES

[1] R. I. Bahar, G. Albera, and S. Manne. Power and performance tradeoffs using various caching strategies. In *Proceedings of the International Symposium on Low-Power Electronics and Design*, 1998.

[2] F. Bellosa. The benefits of event-driven energy accounting in power-sensitive systems. In *Proceedings of 9th ACM SIGOPS European Workshop*, September 2000.

[3] R. Berrendorf and B. Mohr. *PCL - The Performance Counter Library: A Common Interface to Access Hardware Performance Counters on Microprocessors (Version 2.0)*. http://www.kfa-juelich.de/zam/PCL/.

[4] D. Brooks and M. Martonosi. Dynamically exploiting narrow width operands to improve processor power and performance. In *Proceedings of the 5th International Symposium on High Performance Computer Architecture*, Jan. 1999.

[5] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *Proceedings of the 27th International Symposium on Computer Architecture*, June 2000.

[6] D. Burger, T. M. Austin, and S. Bennett. Evaluating future microprocessors: the SimpleScalar tool set. Tech. Report TR-1308, Univ. of Wisconsin-Madison Computer Sciences Dept., July 1996.

[7] T. Cignetti, K. Komarov, and C. Ellis. Energy estimation tools for the Palm^{TM}. In *ACM - MSWiM 2000: Modeling, Analysis and Simulation of Wireless and Mobile Systems*, August 2000.

[8] D. Folegnani and A. Gonzalez. Reducing power consumption of the issue logic. In *Proceedings 2000 Workshop on Complexity Effective Design (WCED) at ISCA2000*, June 2000.

[9] Intel Corporation. *Overview of the VTune^{TM} Performance Analyzer.* http://developer.intel.com/software/products/vtune/.

[10] T. C. Lee, V. Tiwari, S. Malik, and M. Fujita. Power analysis and low-power scheduling techniques for embedded DSP software. *IEEE Transactions on VLSI Systems*, December 1996.

[11] S. Manne, A. Klauser, and D. Grunwald. Pipeline gating: Speculation control for energy reduction. In *Proceedings of the 25th International Symposium on Computer Architecture*, pages 132–41, June 1998.

[12] T. Mathisen. Pentium secrets. *Byte Magazine*, pages 191–192, July 1994.

[13] A. Rogers, M. Carlisle, J. Reppy, and L. Hendren. Supporting dynamic data structures on distributed memory machines. In *ACM Transactions on Programming Languages and Systems, 17(2)*, March 1995.

[14] J. Russell and M. Jacome. Software power estimation and optimization for high performance, 32-bit embedded processors. In *Proceedings of the International Conference on Computer Design*, October 1998.

[15] H. Sanchez et al. Thermal management system for high performance PowerPC microprocessors. In *Proceedings of CompCon '97*, Feb. 1997.

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

[17] V. Tiwari, S. Malik, A. Wolfe, and M. Lee. Instruction level power analysis and optimization of software. *Journal of VLSI Signal Processing*, 13(3):223–238, September 1996.

[18] M. Toburen, T. Conte, and M. Reilly. Instruction scheduling for low power dissipation in high performance microprocessors. Technical report, North Carolina State University, May 1998.

[19] N. Vijaykrishnan. Energy-driven integrated hardware-software optimizations using SimplePower. In *Proceedings of the 27th International Symposium on Computer Architecture*, 2000.

[20] M. A. Viredaz and D. A. Wallach. Power evaluation of Itsy version 2.3. Tech. Note TN 57, Digital Western Research Laboratory, October 2000.

[21] V. Zyuban and P. Kogge. The energy complexity of register files. In *Proceedings of the International Symposium on Low-Power Electronics and Design*, pages 305–310, 1998.