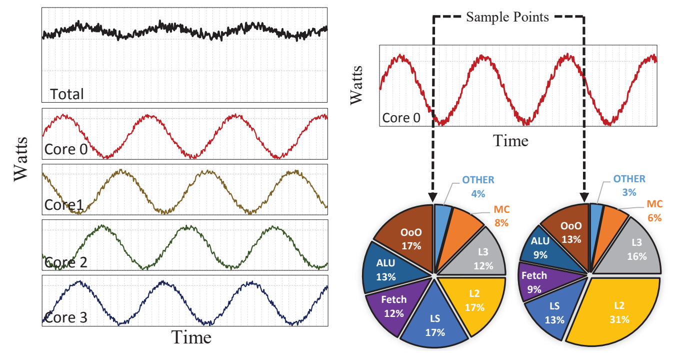
WattWatcher: Fine-Grained Power Estimation on Live Multicore Systems Using Configurable Models

Michael LeBeane*, Jee Ho Ryoo†, Reena Panda‡, and Lizy K. John§
Department of Electrical and Computer Engineering, The University of Texas at Austin
Email: *mlebeane@utexas.edu, †jr45842@utexas.edu, ‡reena.panda@utexas.edu , §ljohn@ece.utexas.edu

***Abstract*—Power is a first-order design constraint in any modern computing platform. Extensive research has focused on estimating power to guide advances in power management schemes, thermal hot spots, and voltage noise. However, most power estimation tools possess drawbacks related to the level of detail and amount of training needed for reliable results. This paper introduces WattWatcher, a power measurement framework that provides fine-grained power traces by passing predefined performance counters and a hardware descriptor file into configurable back-end power models. We show that WattWatcher has a mean absolute percentage error of 2.67% aggregated over all benchmarks when compared to measured power consumption on SPEC CPU 2006 and PARSEC benchmarks across three different machines.**



(a) Per Core Power Breakdown (b) Functional Unit Power Breakdown

Fig. 1. Illustration of important fine-grained monitoring features provided by WattWatcher.

I. Introduction

Estimating the power and energy consumption of processors, either at runtime or offline, is a critical concern in modern machines. Currently, coarse grained measurements can be obtained through hardware power counters [1] [2] or external probes [3]. While processor wide power metering may be useful for high level policies, such aggregate studies mask important internal power consumption trends. Figure 1(a) illustrates a common scenario where individual core power consumption trends drastically differ from the power consumption of the entire processor. Isolating and understanding the trends in individual core power consumption is critical for researching and applying per core DVFS, power capping, and power gating techniques. Even core level power estimates, however, may prove insufficient for certain areas of active research. Figure 1(b) illustrates how functional unit power consumption can vary drastically over the course of program execution. Measuring the power consumption of the internal components of a core is critical for studies in thermal analysis and voltage noise.

Some curve fitting solutions [4][5][6][1] can overcome the granularity limitations inherent in coarse grained power probes, but require extensive training on numerous targeted microbenchmarks, limiting their deployability. These solutions are also extremely sensitive to the coverage of the training set, and fail when exposed to trends that they have not yet encountered. In this paper, we present WattWatcher, a methodology and accompanying tool-kit that uses detailed configurable models from the computer architecture simulation domain and adapts them for power modeling on real live multicore systems. WattWatcher works by collecting performance events from the system under test (SUT) and passing them through easily customizable power models. Our work offers several contributions over the prior art, and carves out a unique and important spot in available power estimation methodologies:

- WattWatcher can model a variety of different processors with its extensible configuration interface. The statistics collected by WattWatcher are generic enough to apply to most modern processors in a variety of form factors. Researchers and vendors can add other processors to our tool by mapping these machines to the WattWatcher interface.
- WattWatcher's power models do not require significant training. Most curve fitting models require an extensive amount of training over a wide enough sample space to cover all possible program types that will be run in the future. WattWatcher specifically models all of the major functional units in a microprocessor by querying the SUT for its hardware configuration.
- WattWatcher offers power breakdowns at the individual core and functional unit granularity. This supports advanced research that requires a much finer level component breakdown than is available from coarse grained monitoring tools.

This paper is organized into several sections that present the WattWatcher methodology and toolkit. Section II explains the design of WattWatcher. Section III evaluates how WattWatcher performs on three different processors. Finally, section IV concludes the paper.

II. WattWatcher: Overview and Operation

In this paper we introduce WattWatcher, a tool that measures performance events in live systems and manipulates them to drive detailed configurable power models. WattWatcher estimates structure and functional unit access patterns to produce input traces suitable for power and timing models traditionally associated with cycle accurate simulation. By calling into the back-end power model at a much larger time granularity than traditional cycle accurate simulators, we can utilize these power models in a realtime environment.

There are a number of configurable power simulators that could serve as the backend for WattWatcher [7][8][9]. For the

TABLE I. GENERIC HARDWARE EVENTS COLLECTED BY WATTWATCHER.

Category	Hardware Events
Core/Kernel	cycles, instructions, uops, migrations
Frontend	branches, mispredictions, IC/iTLB misses
LS	L1/L2/LLC misses, LLC/dTLB misses
Execution	FP scalar, FP packed

specific embodiment of the tool presented in this paper, we have chosen to implement WattWatcher around McPAT [8], due to its frequent updates, verification, and popularity within the computer architecture community. The following sections describe how WattWatcher integrates real hardware information into the back-end model and the layout of the tool.

A. Modeling

Since most configurable architectural power models are originally designed for simulation environments, they require a large number of input statistics representing precise events inside the microarchitecture. However, researchers have noted a tight correlation between power consumption and only a few important hardware events that are exposed as performance counters [4]. WattWatcher leverages this knowledge by monitoring only the generally available counters shown in Table I, deriving the inputs to the power model either directly from these counters, or by making reasonable estimations and transformations. A few simple examples are illustrated in the following paragraphs.

OoO Engine and Instruction Fetch: These structures include the reorder buffer, instruction window, and reservation stations. Accesses to these data structures are estimated directly based on the instructions (either RISC instructions or x86 micro ops) issued and retired.

Register File Accesses: Register file access are estimated based on the number and type of instructions issued. For example, we assume that integer instructions will perform on average two reads to the integer register file, and one write. Similarly for floating point, where the width of the read and the number of instructions is determined by characteristics of the instruction (single vs. double precision and packed vs. scalar).

Memory Hierarchy Accesses: Most memory hierarchy statistics are collected directly from the processor counters. Load and store queue activity is based on the L1 D-cache statistics. Raw number of access to a level of the memory hierarchy are estimated based on the number of misses in the level directly above it. Prefetches are counted separately.

Although WattWatcher focuses on modeling fine-grained functional unit power variations, it can also model states exposed through ACPI. Performance states (P-states) can be modeled by feeding operating voltage and frequency into the configurable back-end power model. For measuring only dynamic power, WattWatcher will work correctly out of the box. Modeling leakage during processor states (C-states) is challenging, however, since the unique combination of when and where to gate certain components is vendor dependent and not generally available to the public. This is further exacerbated by McPAT's inaccurate modeling of leakage in devices using modern manufacturing technologies [10]. Fortunately, processor leakage is relatively constant during each operating processor state and can be accommodated for in one of two ways. First, WattWatcher can be calibrated by measuring leakage power using a coarse grained measurement tool, such as RAPL or hardware probes, at each available processor state. WattWatcher will use this information along with statistics it gathers on core C-state transfers to factor in the correct leakage power. Secondly, if no coarse grained measurement tool is available, the datasheet for the processor can be consulted for ideal numbers regarding power dissipation in different C-states.

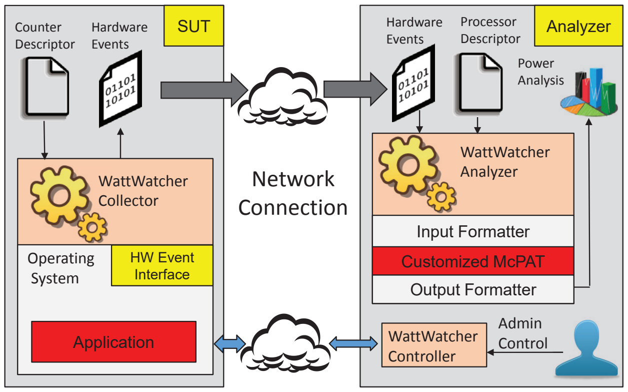


Fig. 2. Overview of the WattWatcher toolchain.

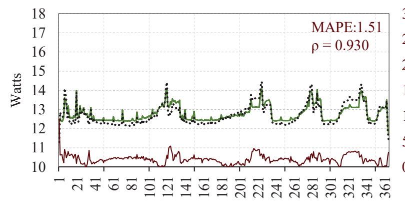
B. Tool Overview

WattWatcher is a toolkit that integrates a number of Linux utilities and McPAT together with configurable system models and functional unit estimators. The toolkit is divided up into three components: the Controller, Analyzer, and Collector. These three elements work together and interact with each other to comprise the full WattWatcher toolkit. A description of how these elements operate together in a common workflow is presented in Figure 2 and is described in the following paragraphs in detail.

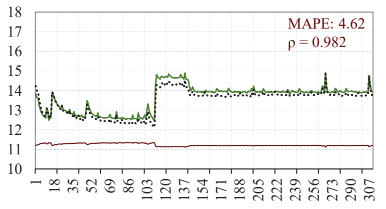
Controller: All user interaction with the WattWatcher system is initiated via the Controller module. The user passes a number of parameters to the Controller at startup, such as the location (hostname) of the SUT(s) and the counter descriptor file containing the umask and event numbers. The WattWatcher Controller opens a connection to the SUT(s) and queries it to gather high level statistics on microarchitectural features such as cache layout, number of CPUs, and core frequency. These statistics are used to populate an XML file that represents the machine configuration in McPAT. Functional unit information is estimated from a pre-populated table of common system configurations. This information can also be overridden by a user's custom configuration file, in the event that automatic discovery is insufficient, or the microarchitecture is very unconventional. The Controller then stores the system configuration for later use and proceeds to launch the Collector with the counter descriptor and machine configuration.

Collector: The Collector is in charge of gathering runtime statistics on the SUT. Towards this end, the Collector uses the popular Linux performance monitoring tool, perf [11]. Perf logs hardware performance counter information at a user defined sampling rate. The counter descriptor file provided by the Controller determines exactly which hardware events to collect, and how to classify them. For live analysis mode, the Collector constantly pipes the data to the Analyzer for immediate processing. For off-line analysis, the data is buffered on the Collector node and sent in bulk at the end of a run.

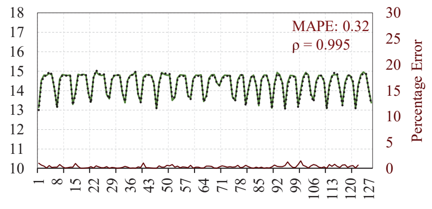
Analyzer: The Analyzer is the main module of the WattWatcher toolchain. It is responsible for turning the raw data transferred by the Collector into power estimates for the



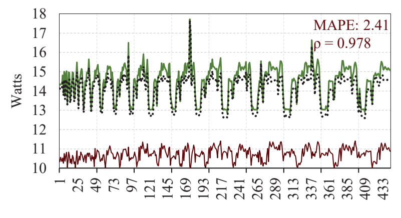
(a) mcf from SPECint



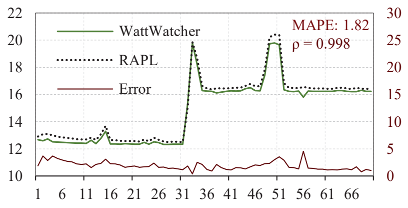
(b) xalancbmk from SPECint



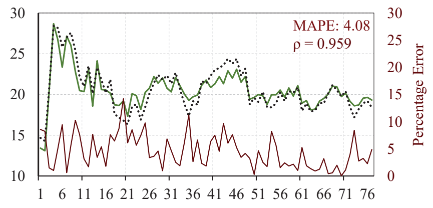
(c) bwaves from SPECfp



(d) dealII from SPECfp



(e) canneal from PARSEC



(f) facesim from PARSEC

Fig. 3. Comparison of WattWatcher to RAPL power counters. The x-axis is the runtime of the benchmark in seconds.

TABLE II. SUTs USED TO EVALUATE THE PROPOSED METHODOLOGY.

Alias	Model	Form Factor	TDP
SUT0	Intel i7 2720QM	Laptop (32nm)	45W
SUT1	Intel i7 4700QM	Laptop (22nm)	47W
SUT2	AMD A10-6800K	Desktop (32nm)	100W

TABLE III. ERROR ACROSS SUTs (MAE/RMSE/MAPE).

	SPECint	SPECfp	PARSEC	TOTAL
SUT0	0.39/0.42/2.78	0.25/0.29/1.75	1.09/1.26/5.65	0.47/0.55/2.85
SUT1	0.41/0.45/2.51	0.36/0.39/2.10	0.85/0.96/4.81	0.51/0.57/2.97
SUT2	1.31/1.64/1.68	1.62/2.13/2.06	2.17/2.83/2.84	1.69/2.19/2.18

SUT. It first takes the raw data format output by the collector and parses it into McPAT compatible format. The raw data is combined with the system configuration obtained by the Controller to produce an XML file that represents the topology and runtime behavior of the SUT. This information can then be passed directly into McPAT for real time data reporting, or archived for offline analysis. Any missing input parameters to McPAT are estimated from the provided statistics. The results of a McPAT run are then output to the directory specified by the user in an easy-to-parse comma separated values (csv) format.

The setup in Figure 2 is simply one example of how WattWatcher can be configured to monitor power on a target system. One can also run the Analyzer and Controller on the SUT, or the Controller and Analyzer can be configured to monitor more than one system on a cluster consisting of homogeneous or heterogeneous machines.

III. VALIDATION

In this section, we validate WattWatcher on processors from three different vendors, form factors, and manufacturing technologies as illustrated in Table II. For our studies, we use the SPEC CPU2006 [12] and PARSEC [13] benchmark suites. SPEC is an industry standard single-threaded performance benchmarking toolkit, and PARSEC is a research benchmark suite for multicore systems. For all benchmarks, we use the largest input size available to guarantee a runtime in excess of several minutes on our fastest machines.

During validation, we do not allow frequency or voltage modulation, as we would like to show that our tool captures small variations in power related to functional unit activity, not coarse grained power management decisions. We do allow C-states that do not change operating frequency or voltage to avoid no-op loops and polling on idle threads. We calibrate leakage power for all processor states as described in section II-A. All references to hardware measured power will refer to the appropriate power monitoring counters in AMD (APM) and Intel (RAPL) machines.

A. Time Variant Analysis

We now illustrate how WattWatcher correctly measures total power over time when compared to RAPL counters on SUT0. All data was sampled once every second, over the complete execution of the program and is presented in Figure 3. The total power contribution of each individual core has been added to equal total processor power consumption. We aggregate across all cores since the RAPL does not allow for finer grained breakdowns.

Starting from the top left and moving clockwise, Figure 3 shows mcf, xalancbmk, bwaves, dealII, canneal, and facesim. There are two benchmarks represented from each of SPECint, SPECfp, and PARSEC. Results are reported in three ways: instantaneous error, MAPE (mean absolute percentage error) and Pearson's correlation coefficient. Instantaneous error represents the absolute value of the error at each sample point and MAPE is simply the sum of the residuals presented as a percentage of RAPL power. Pearson's correlation coefficient indicates how well WattWatcher tracks RAPL, with 1 indicating a perfect correlation, and 0 indicating no correlation.

WattWatcher correlates with RAPL counters extremely well, with all correlation coefficients greater than 0.9. Figure 3(c) in particular is almost perfectly captured by WattWatcher. From the other figures, we can see that there are two primary sources of error in WattWatcher estimation. The first involves WattWatcher under/over estimating the entire workload by a small constant value. This is illustrated by the sources of error in Figures 3(b), 3(e), and 3(d). The second source of error springs from rapid changes in power. While the upwards and downward trends are almost always captured correctly, the raw magnitude of power spikes is occasionally incorrect. This can be seen most clearly in Figures 3(f) and 3(a). Despite these small inaccuracies, however,

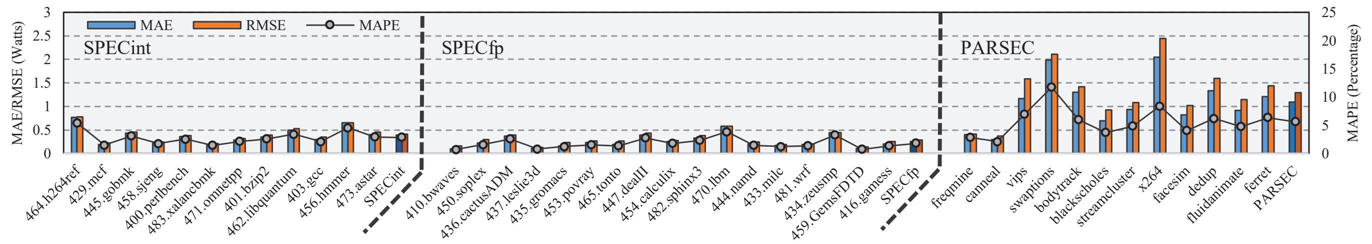


Fig. 4. This figure illustrates the error for each workload when compared to hardware power measurements.

WattWatcher trends very well and the total error for these workloads is less than 5%.

B. Aggregate Analysis

Figure 4 illustrates the accuracy of WattWatcher summed over the duration of each program execution on SUT0. The results are presented as MAE (mean absolute error), RMSE (root mean squared error), and MAPE (mean absolute percentage error). For the SPECint workloads WattWatcher achieves a MAE of 0.39 W, RMSE of 0.42 W, and MAPE of 2.78%. SPECfp workloads exhibit slightly better accuracy due to their repetitive and periodic nature, with a MAE of 0.25 W, RMSE of 0.29 W, and MAPE 1.75%. PARSEC workloads exhibit a very different power profile than their counterparts in SPECint and SPECfp. These workloads are highly multithreaded, and display a great deal of variance when compared with the single threaded workloads, and within themselves. WattWatcher achieves a MAE of 1.09 W, RMSE of 1.26 W, and MAPE of 5.65% for the multithreaded PARSEC workloads. PARSEC's error is generally higher than SPEC's due to the presence of multiple active cores. For SPEC, only one of the cores is active at a time, and the other three cores only use leakage power, which is much easier to estimate. Overall, the MAPE across all workloads on this SUT is 2.85%.

We have also verified WattWatcher over two other SUTs, but only perform a deep dive into SUT0 due to space constraints. Table III shows the total MAPE of WattWatcher over the three SUTs that we tested. The two Intel machines (SUT0 and SUT1) both display similar trends in accuracy across the three categories of programs. The AMD machine (SUT2) displayed less application dependent variations in power than the other two machines, resulting in similar error rates among the programs. We anticipate that this is primarily due to the differences in form factor (laptop vs desktop) rather than an intrinsic difference in the vendors themselves.

We cannot verify the functional unit power consumption in our evaluation, due to the difficulty of measuring real hardware power for the individual functional units in isolation. While it is true that an over-estimation in one component and an under-estimation in another could lead to correct overall trends, we believe that our consistent high accuracy against the RAPL counters minimizes this possibility. Furthermore, our underlying power model, McPAT, has been validated for accuracy at the functional unit level against several RTL models of real hardware in the prior work [8][10].

IV. Conclusion

Understanding power consumption is a critical concern in any modern computing platform. Towards this end, we have developed the WattWatcher methodology and framework for fine-grained power estimation on multicore processors. Our framework estimates power by passing predefined performance counters and a hardware descriptor file into the research standard multicore power model, McPAT. Since McPAT explicitly models each component and functional unit of a modern multicore processor, WattWatcher does not require training like curve fitting power models, and can offer the user a complete breakdown at the functional unit level. WattWatcher is capable of automatically reading the system configuration from a host machine to calibrate the power estimation tool, and can successfully sample power at a minimal sampling period of 100 ms with minimal overhead. We show that WattWatcher has a MAPE of 2.67% of measured power consumption when compared to hardware power on SPEC CPU 2006 and PARSEC benchmarks aggregated across three different machines of differing vendors, form factors, and manufacturing process. Through the use of this methodology, it is possible to obtain a detailed power breakdown on real hardware without vendor proprietary models or hardware instrumentation.

References

[1] R. Joseph and M. Martonosi, "Run-time power estimation in high performance microprocessors," in *ISLPED*, 2001, pp. 135–140.

[2] "AMD BIOS and Kernel Developer's Guide for AMD family 15h Models 00h-0Fh Processors," http://support.amd.com/TechDocs/.

[3] R. Ge *et al.*, "Powerpack: Energy profiling and analysis of high-performance systems and applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 5, pp. 658–671, May 2010.

[4] W. Bircher and L. John, "Complete system power estimation: A trickle-down approach based on performance events," in *ISPASS*, April 2007, pp. 158–168.

[5] S. Gurumurthi *et al.*, "Using complete machine simulation for software power estimation: The softwatt approach," in *HPCA*, 2002, pp. 141–.

[6] C. Isci and M. Martonosi, "Runtime power monitoring in high-end processors: methodology and empirical data," in *MICRO*, Dec 2003, pp. 93–104.

[7] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: a framework for architectural-level power analysis and optimizations," in *ISCA*, June 2000, pp. 83–94.

[8] S. Li *et al.*, "Mcpat: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *MICRO*, Dec 2009, pp. 469–480.

[9] D. Brooks *et al.*, "Power-performance modeling and tradeoff analysis for a high end microprocessor," in *Power-Aware Computer Systems*, 2001, pp. 126–136.

[10] S. Xi *et al.*, "Quantifying sources of error in mcpat and their potential impacts on architectural studies," in *HPCA*. IEEE Computer Society, 2015.

[11] "perf: Linux profiling with performance counters," https://perf.wiki.kernel.org/.

[12] J. L. Henning, "Spec cpu2006 benchmark descriptions," *SIGARCH Comput. Archit. News*, vol. 34, no. 4, pp. 1–17, Sep 2006.

[13] C. Bienia, "Benchmarking modern multiprocessors," Ph.D. dissertation, Princeton University, January 2011.