2015 27th International Symposium on Computer Architecture and High Performance Computing

WattWatcher: Fine-Grained Power Estimation For Emerging Workloads

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

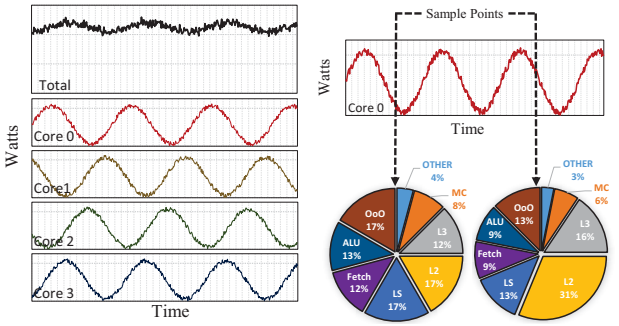
***Abstract*—Extensive research has focused on estimating power to guide advances in power management schemes, thermal hot spots, and voltage noise. However, simulated power models are slow and struggle with deep software stacks, while direct measurements are typically coarse-grained. This paper introduces WattWatcher, a multicore power measurement framework that offers fine-grained functional unit breakdowns. WattWatcher operates by passing event counts and a hardware descriptor file into configurable back-end power models based on McPAT. Researchers and vendors can add other processors to our tool by mapping to the WattWatcher interface. We show that WattWatcher, when calibrated, has a MAPE (mean absolute percentage error) of 2.67% aggregated over all benchmarks when compared to measured power consumption on SPEC CPU 2006 and multithreaded PARSEC benchmarks across three different machines of various form factors and manufacturing processes. We present two use cases showing how WattWatcher can derive insights that are difficult to obtain through other measurement infrastructures. Additionally, we illustrate how WattWatcher can be used to provide insights into challenging big data and cloud workloads on a server CPU. Through the use of WattWatcher, it is possible to obtain a detailed power breakdown on real hardware without vendor proprietary models or hardware instrumentation.**

I. Introduction

Estimating power and energy consumption of processors is a critical concern on modern machines. Currently, coarse grained measurements can be obtained through hardware power counters or external probes. However, emerging workloads, such as big data applications, typically contain a mix of functional-unit and thread-level interactions. While processor wide power metering may be useful for high level policies, coarse grained approaches often mask important internal power consumption trends.

Some examples of modern trends that can be difficult to observe with most current power monitoring tools are shown in Figure 1. In Figure 1(a), the total power envelope differs substantially from the power consumption of each individual cores. This is an increasingly common trend in complex applications with significant OS interaction, or multi-node applications that separate network progress threads from the application itself. Isolating and understanding per core consumption is critical for researching and applying per core DVFS, power capping, and power gating techniques. Figure 1(b) illustrates another example where the proportion of dynamic power spent by each of the major functional units changes as the application executes through different phases. Correctly capturing these fine-grained variations is critical for studies in thermal analysis and voltage noise.

The most popular fine-grained power estimation technique, cycle accurate simulation, can offer a high degree detail. However, simulation is extremely slow and not used to estimate power in any real-time environment. Additionally, cycle accurate simulation frequently struggles with the complex software stacks inherent to emerging workloads. While simulators are becoming more and more robust, most still struggle with virtual-machine based languages like Java which comprise the heart of many big data stacks. Many more still do not factor in the complex interplay of kernel code and context switching between OS managed threads and application programs.



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

Fig. 1. Illustration of important fine-grained monitoring features.

In this paper, we present WattWatcher, a real-time power monitoring framework that estimates power on live systems and can deliver the level of detail illustrated in Figure 1 for any workload. WattWatcher is a methodology and accompanying tool-kit that uses detailed configurable models from the computer architecture simulation domain and adapts them for power modeling on live multicore systems. The WattWatcher toolkit 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:

1) 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.

2) WattWatcher's power models require only a small amount of coarse grained calibrations at important p-states and c-states. Most curve fitting and learning based 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 avoids training by specifically modeling all of the major functional units in a microprocessor using McPAT.

3) WattWatcher offers power breakdowns at the individual core and functional unit granularity. This supports advanced

1550-6533/15 $31.00 © 2015 IEEE
DOI 10.1109/SBAC-PAD.2015.26

TABLE I. RELATIVE CLASSIFICATION OF PRIOR WORK AND WATTWATCHER.

Approach	Accuracy	Detail	Frequency	Cost	Speed
Direct Measurement	++	-	us-ms	-	Fast
Power PMCs	+	-	ms	=	Fast
Curve Fitting	=	=	us-s	+	Offline
Simulators	+	+	ns	+	Slow
WattWatcher	+	+	ms	+	Fast

research that requires a finer level component breakdown than is available from coarse grained monitoring tools.

Throughout this paper, we illustrate a robust validation of WattWatcher over a three different SUTs of various vendors and form factors. We show that WattWatcher achieves a 2.67% MAPE (mean average percentage error) when compared to the highly accurate RAPL coarse grain monitoring hardware. Additionally, we present several case studies on traditional and emerging workloads. These case studies illustrate how WattWatcher can show microarchitectural trends that are missed by prevalent power monitoring tools.

This paper is organized into several sections that present the WattWatcher methodology and toolkit. Section II surveys the prior art and explains the benefits and limitations of other approaches. Section III explains the design of WattWatcher. Section IV evaluates how WattWatcher performs on three different processors. Section V presents two example use cases and a power study of difficult to simulate big data workloads. Finally, section VI concludes the paper.

II. BACKGROUND

A great deal of work has gone into estimating processor power, both within the research community and by industry professionals. Table I breaks down the approaches used by previous researchers and computing professionals into several different categories, each with their own strengths and weaknesses. Existing approaches are classified using the following relative scale, where +/++ indicates an advantage, = is neutral, and - indicates a disadvantage.

Direct Measurements: A commonly utilized methodology in the field for power measurement is to install an external analog power probe directly to the SUT [1]. While these devices can be highly accurate depending on the quality of the probe, there are a number of drawbacks. First, the level of detail from wall probe measurements is very coarse and reflects the entire power consumption of all devices that draw energy from that outlet. More fine grained isolation of components requires destructive shunting of potentially very high current circuits. Additionally, isolating the power consumption at a functional unit level in the core is not possible using external tools. Finally, scale-out deployments would require a probe on every machine, driving up the cost of buying already expensive monitoring equipment.

Curve Fitting: The methodology most prevalent in academia correlates performance counters to power using curve fitting and machine learning models [2][3][4][5][6]. Carefully designed and calibrated regressions can discover correlations between performance events and power. While these methodologies are often effective, they require extensive training and calibration. A poor training set can lead to incorrect results. Training also implicitly ties these models to a particular microarchitecture and limits deployability, since a SUT must already have a way of measuring power before training can occur.

TABLE II. EVENTS COLLECTED BY WATTWATCHER

Category	Hardware Events
General	Context Switches, Frequency, Voltage, Cycles
Frontend	Branch Mispredictions, IC Misses, iTLB Misses, uops Issued
LS/Caches	L1 Misses/Hits, L2 Misses, LLC Misses, dTLB Misses
Execution	FP Scalar, FP Packed, FP Width
Retirement	uops Retired

Power Performance Monitoring Counters (PMCs): Intel's Running Average Power Limit (RAPL) [6] and AMD's Application Power Management (APM) [7] frameworks now provide performance counters that can estimate power consumption on a target platform. Although these counters were designed primarily to perform power capping, they can also be used as a generic metering device. Intel's embodiment of this concept has been verified by the research community [8] and is widely accepted as an accurate power measurement framework with very low overhead. Unfortunately, they are not useful for fine grained analysis, since they only report power at the package or aggregate processor core granularity.

Power Simulators: A number of low level power models exist as part of a particular architectural simulator, or that can be plugged into a generic performance simulator [9][10][11][12]. These power models extract detailed functional-unit level accesses from their respective performance simulators to provide detailed statistics at any required time resolution. However, power models that rely on performance simulators are limited by the traditional problems associated with microprocessor simulation, such as extremely slow runtime, simulator bugs, and incompatibility with certain programs.

III. WATTWATCHER: OVERVIEW AND OPERATION

In this paper we introduce WattWatcher, a tool that measures and modifies performance events in live systems 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 [9][13][10][12]. For the specific embodiment of the tool presented in this paper, we have chosen to implement WattWatcher around McPAT [10], due to its frequent updates, verification, and popularity within the computer architecture community. In McPAT, the individual functional units and caches modeled by CACTI [11] are combined into a complete multicore processor model. 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 [2]. Generally, dynamic power consumption is largely dependent on overall machine activity, which can be expressed by active cycles and instructions. Additionally, many events in a microarchitecture are tightly correlated, such as LLC

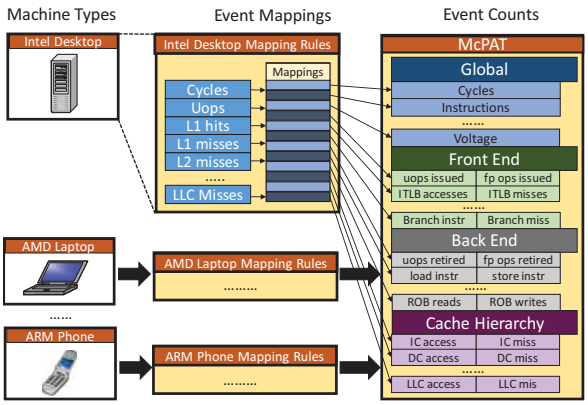


Fig. 2. WattWatcher mapping layer translates machine dependent counters into general purpose McPAT events.

misses/prefetches and memory controller activity, allowing power for some functional units to be estimated without an explicit counter.

WattWatcher leverages this knowledge by using relatively few counters to estimate a large number of statistics needed for McPAT's power model. Keeping the required counters small and general purpose reduces error related to counter multiplexing and increases compatibility of WattWatcher accross many different microarchitectures and processor types. Figure 2 illustrates WattWatcher's transformation of raw counter data into McPAT compatible statistics. Essentially, every microarchitecture requires a mapping file that defines the relationship between a microarchitecture's performance counters and the backend McPAT event counts. These mapping files contain a number of mapping rules that combine and manipulate counters into detailed McPAT events. Some of these events, such as cycles and instructions, can be directly estimated from generally available performance counters. Others, such as the number of reads and writes to an ROB for out-of-order processors, requires some reasonable assumptions and estimations. To provide transparency into the mapping process, we show some selected samples from a typical x86 OoO microarchitecture in Table II. Additionally, we show how the mapping rules convert the counters to McPAT events. Identity rules are omitted for brevity.

OoO Engine Statistics: These structures include the reorder buffer, instruction window, and reservation stations. McPAT requires read and write access counts to each one of these structures. Accesses related to the instruction window and reservation stations are estimated directly based on the x86 uops issued. Access related to the RoB and common data buses on the backend are estimated based on the number of uops retired. The difference between the issued and retired uops are used to determine write accesses to components that commit microarchitectural state.

Register File Statistics Register file access are estimated based on the number and type of instructions issued. We differentiate between integer and floating point register files. While there is an explicit accounting for floating point uops, there is no corresponding counter for integer instructions. Therefore, we classify all instructions other than floating point as integer. We then 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

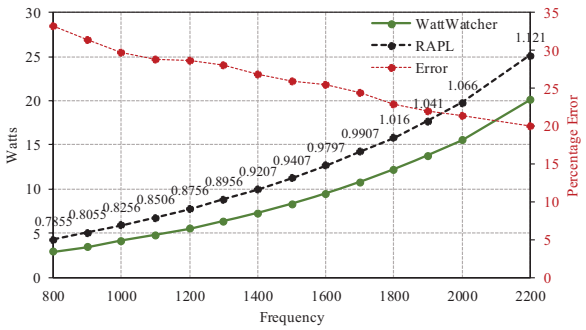


Fig. 3. WattWatcher P-State (DVFS) modeling on on SUT0. Line labels indicate the voltage for the performance state frequency on the x-axis.

characteristics of the instruction (single vs. double precision and packed vs. scalar). This model is most likely an overcount of the number of register file accesses. However, a similar accounting of register file accesses has been performed before and found to be a reasonable approximation [14].

Memory Hierarchy Statistics For memory hierarchy accesses, we rely heavily on a 'trickle-down' approach [2]. For demand caching events, we explicitly track misses at all levels of the cache, but do not count accesses accept for the L1 caches. Access to all further caches in the hierarchy are estimated based on miss events from the higher level. This technique is used on all cache levels and extends to the integrated memory control. Such a technique is a good approximation when cache line sizes are consistent, but are less accurate when cache line sizes vary or there is a significant amount of request combining at lower levels of the hierarchy. Prefetches for levels of the cache which support them are counted separately using the appropriate counters, since these are difficult to estimate otherwise.

Duty Cycle In addition to raw event counts, WattWatcher estimates the duty cycle of the functional units in the machine. The instructions issued to each functional unit is divided by the number of cycles in the time interval to obtain the duty cycle. For instructions such as floating point or multiply, the instruction is weighted by the latency of the functional unit as provided by the appropriate processor documentation. This is to ensure an appropriate duty cycle for these components in the presence of heavy pipelining.

Not all of the events used for this example OoO processor will apply to other processor types. Fortunately, McPAT is capable of modeling processors of various different types, from massive out of order execution engines, to more conservative in-order designs. All that is required is a different machine configuration and corresponding mapping file. As part of this work, we have developed mapping functions for an AMD Piledriver, Intel Haswell, Intel Sandy Bridge, and an older Intel Penryn microarchitecture.

B. Calibration

Although WattWatcher focuses on modeling fine-grained functional unit power variations, it can also model power states exposed through ACPI. While WattWatcher will work out of the box, the underlying McPAT performance modeling has been found to consistently underestimate the power of the processor[10]. In order to counteract these effects, WattWatcher employs a small amount of calibration at different operating points using either RAPL or hardware probes in the

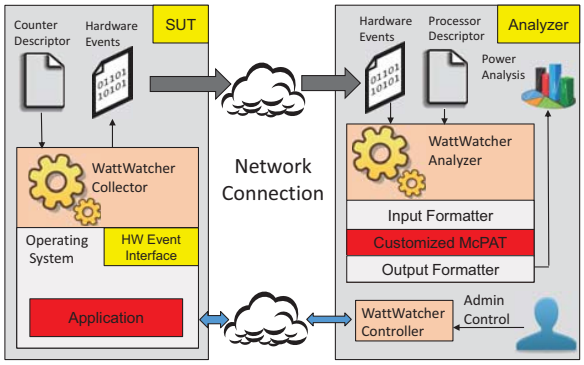


Fig. 4. Overview of the WattWatcher toolchain.

system. The baseline power results collected by WattWatcher are multiplicatively scaled based on the cores p-states/c-states using the following calibration methodology.

For performance states (P-states), we force the processor into an no-op loop at each available p-state and measure the power consumed on the system. We then feed in the voltage id and operating frequency from the machine status register into the WattWatcher tool. As an example, Figure 3 shows the DVFS calibration phase for one of our test machines over all available performance states. The points on the lines correspond to available P-states in the system and the labels correspond to the operating voltage at that P-state. These results indicate that uncalibrated WattWatcher correctly models the power trends in the curve, but consistently underestimates the magnitude by approximately 25%. This is in accordance with the numbers reported by McPAT [10] for Intel processors. Additionally we have found that the accuracy of the backend McPAT model increases as the voltage and frequency are increased, a trend reflected over all machines we studied. WattWatcher uses the error generated by this one-time calibration to scale future results to obtain the correct power.

For idle power states, WattWatcher employs a similar calibration technique to determine power at different C-states. However, it does not attempt to calibrate itself to the measured values. Instead, it simply reports the calibrated value when it detects a core transition into an idle state. The unique combination of power gating and clock gating used during idle states is highly vendor dependent, and cannot be adequately described using a general purpose power modeling tool like WattWatcher. This is acceptable since there is extremely little variation in power dissipation while a core is in an idle state.

If no coarse grained measurement tool is available in the system, then WattWatcher can be calibrated using static estimates. One reasonable static calibration method would involve utilizing the numerous studies comparing McPAT to commercial processors performed by other researchers [15][10] to scale the output of WattWatcher. Another C-state specific calibration technique could use the datasheet optimal C-state dissipation numbers for the processor in question [16]. Finally, if only the relative difference in power consumption over time is needed by the end user, then WattWatcher can skip calibration and be used out of the box.

C. 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 4 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.

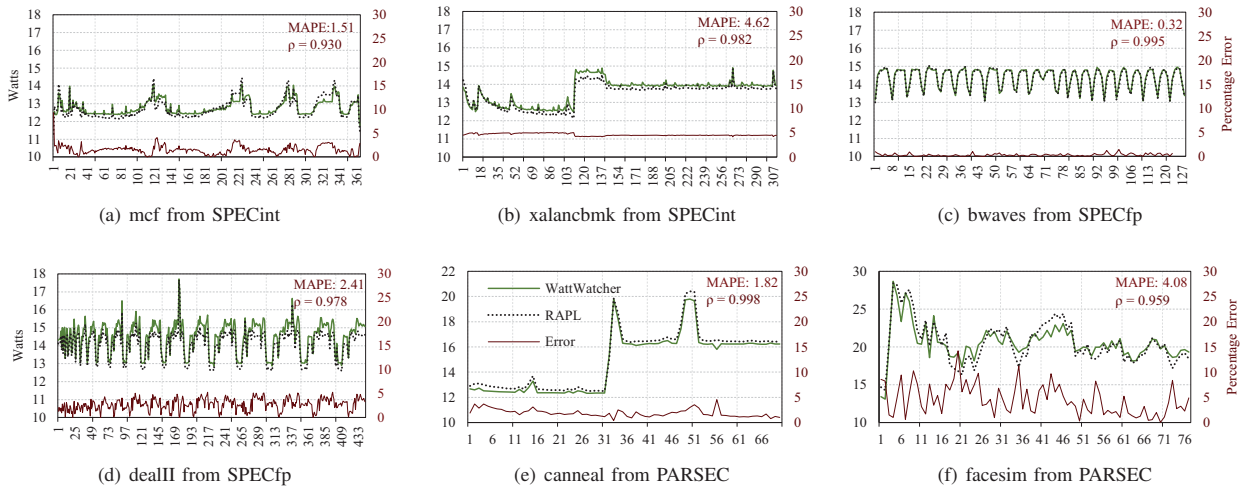
Collector: The Collector is in charge of gathering runtime statistics on the SUT. Towards this end, the Collector uses the popular Linux performance monitoring tool, perf [17]. Perf logs hardware performance counter information at a user defined sampling rate. The counter descriptor file provided by 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 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 4 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.

D. Overheads and Limitations

While WattWatcher fills an important role in power estimation frameworks, it is not a universal panacea. The limitations of the toolkit must be understood in order to make sure it is applied in appropriate contexts. The most fundamental limitation of WattWatcher is that it inherits the inaccuracies of the McPAT model which it is based upon. As such, WattWatcher does



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

require some calibration to optimally monitor coarse grained power fluctuations imposed by P-state and C-state transitions. Additionally, WattWatcher does not currently model or detect changes in power caused by chip aging or environmental effects. Finally, "hidden" coarse grained power states not visible to the OS will not be detected by the WattWatcher system.

Like any real-time analysis tool, there are overheads associated with running WattWatcher. In a configuration similar to that shown in Figure 4, the overheads on the SUT are only from the Collector. The main computational overhead of the Collector is from accessing the perf userspace tools. Perf related overheads are well understood and quantified [18], measured in units of thousands of cycles per event access. Since the tool limits the sampling rate to 100ms, the overhead of querying even a large number of performance counters in that time period is less than 1% on a machine with a core clock of at least 100MHz.

For configurations where all the components are run on the SUT, the overhead is higher. While the Controller overhead is negligible, the Analyzer module must run the McPAT program for every sample point. To accelerate McPAT, we have adopted a scheme where the individual CACTI models are stored in a database [19] the first time they are calculated. Every subsequent request for the same CACTI model will be drawn from this database instead of being recomputed, similar to [20]. Regardless, we recommend reserving one core of the system to handle the overhead of the analysis module. Reserving a core also offers the additional advantage that the energy consumption of the toolkit itself can be ignored by configuring the collector to ignore the monitoring core. For server processors this is an effective solution, as future trends point to more and more cores per socket. For embedded applications, we strongly recommend separating the toolkit as shown in Figure 4.

IV. Validation

In this section, we validate WattWatcher on processors from three different vendors, form factors, and manufacturing technologies as illustrated in Table III. For our studies, we use

TABLE III. 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

the SPEC CPU2006 [21] and PARSEC [22] 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 allow C-states that do not change operating frequency or voltage to avoid no-op loops and polling on idle threads. We calibrate coarse grained power for all processor performance and idle states as described in section III-B. 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. For subsequent 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. The results have been scaled in accordance with the DVFS study so that we can identify fine-grained errors. All data was sampled once every second, over the complete execution of the program and is presented in Figure 5. 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, Figure 5 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

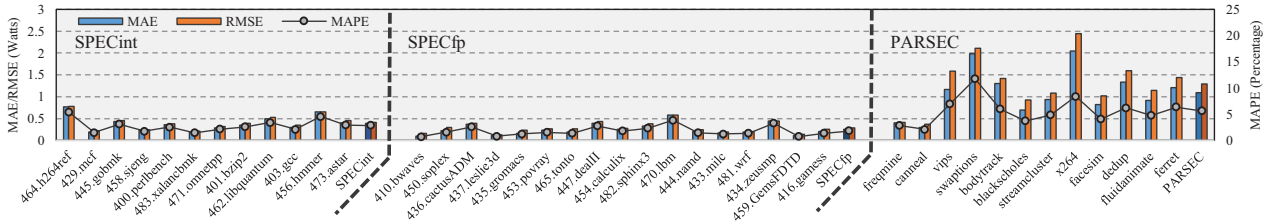


Fig. 6. This figure illustrates the error for each workload when compared to hardware power measurements.

TABLE IV. ERROR ACROSS SUTS: MAE(W)/RMSE(W)/MAPE(%W).

	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

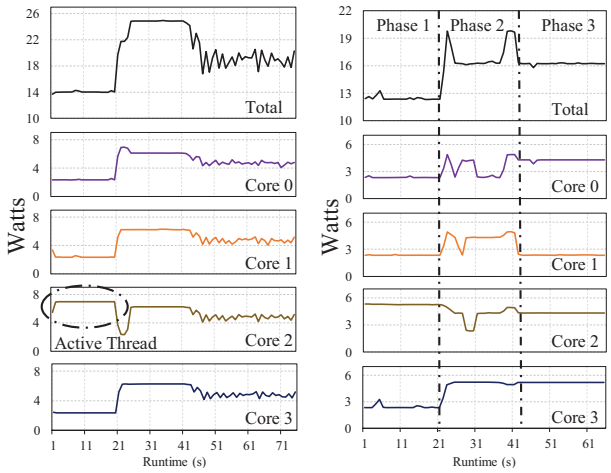
RAPL power. Pearson's correlation coefficient indicates how well WattWatcher tracks RAPL, with 1 indicating a perfect correlation, and 0 indicating no correlation.

WattWatcher correlates with RAPL counters extremely well, with all correlation coefficients greater than 0.9. Figure 5(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 5(b), 5(e), and 5(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 5(f) and 5(a). Despite these small inaccuracies, however, WattWatcher trends very well and the total error for these workloads is less than 5%.

B. Aggregate Analysis

Figure 6 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 IV 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.



(a) Core power for blackscholes (b) Core power for canneal

Fig. 7. Core-wise breakdown for two PARSEC workloads.

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 [10][15].

V. CASE STUDIES

We will now illustrate an important aspect of the WattWatcher framework; the ability to monitor individual processor components on live systems. Understanding the power consumption of each component within the core is an essential feature for researchers. Identifying power hungry and idle areas and transitions within a core can help provide insights into the design of highly aggressive clock and power gating strategies. Additionally, a proper component-wise power profile allows for accurate thermal and voltage noise modeling using any number of available tools [23][24] that accept detailed power traces.

A. Core Breakdown

WattWatcher is capable of isolating the power consumption of individual cores on a processor package. While RAPL coun-

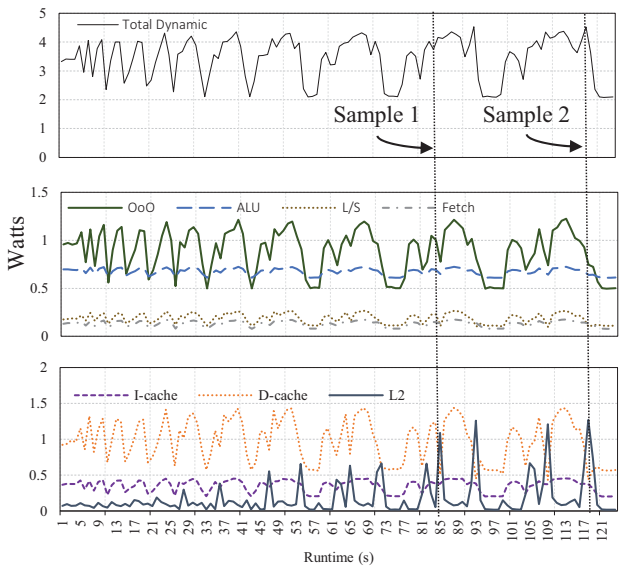


Fig. 8. Functional unit breakdown of dealII.

ters and external probes can provide a total estimation of aggregate power, they do not provide a per core breakdown. Such a breakdown is useful for per core power management strategies and research. Figure 7 illustrates the blackscholes and canneal PARSEC workloads profiled at a core level by WattWatcher. In Figure 7(a), The individual cores track the trends of the processors power consumption, except for the area marked as 'Active Thread'. This represents the workload setup phase in PARSEC, which is single threaded. Figure 7(b) illustrates the core-wise power breakdown of canneal, which has been broken down into three phases. Phase 1 is the workload setup stage, which is single threaded as in blackscholes. For Phase 2, all cores on the machine are active, but vary greatly in their activity level as threads are spawned, destroyed, and migrated between processor cores. Finally, Phase 3 illustrates relatively constant power consumption among all cores except core 1, which is idle. WattWatcher reveals these differences in core behavior, which is masked when using tools that force aggregation of power.

B. Functional Unit Breakdown

WattWatcher can also analyze individual cores at the level of functional units. Figure 8 presents dynamic power sampled every one second from the single active core of the dealII workload. The total power consumption illustrates a periodic trend with a few spikes in the upper plateau. However, the aggregate trend masks the intricacies of each functional unit. The individual functional unit breakdown exhibits significant variation in dynamic power, with no one functional unit perfectly representing the trends in aggregate power. The most interesting interaction in this workload concerns the L2 cache, OoO engine and the D-Cache. During the instances marked 'Sample 1' and 'Sample 2' the OoO and D-cache experience a drop in dynamic power consumption, while the L2 cache spikes dramatically. In 'Sample 1', this trend is completely masked by looking at total power, since the drop in OoO and D-cache power is almost the same as the increase in L2 power. In 'Sample 2', the L2 cache causes a spike that appears in aggregate power. Neither one of these trends would be visible

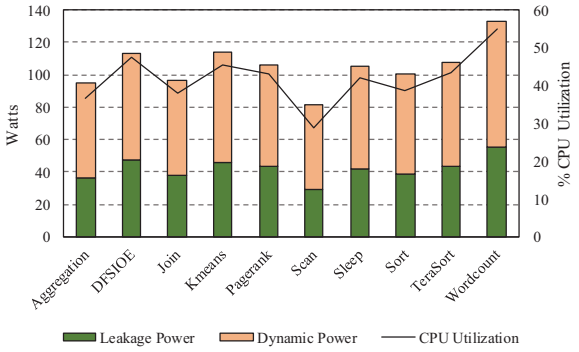


Fig. 9. Dynamic and leakage power consumed by HiBench MapReduce workloads.

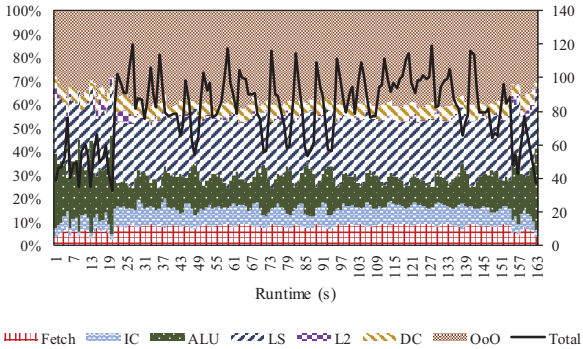


Fig. 10. Dynamic Power Consumption of WordCount benchmark.

by simply looking at aggregate power metering such as those provided by RAPL, but are visible using WattWatcher.

C. Big Data Workloads

One major benefit of the WattWatcher framework is that it allows for the analysis of large, multi-node and emerging workloads that are extremely difficult to simulate in detail. One example of such a workload are the important big data and cloud programs. These workloads often operate over multiple nodes and require a large software stack. In this section, we will show how WattWatcher can be used to explore the power consumption of this class of workload. For this study, we will focus on the popular MapReduce [25] programming paradigm. Our workloads are selected from the HiBench suite [26] and are run on a multi-socket Xeon E5405 server platform supporting idle states. Each socket has a TDP of 80W.

Figure 9 shows the power consumption averaged across the execution of each workload broken down into leakage and dynamic power. We see from the figure that power consumption varies significantly among workloads, with approximately 50W swing (25W per socket) between the most and least power hungry workloads. To understand why there is such variation in power, we superimpose a line illustrating CPU utilization onto the graph. For these Hadoop based workload, there is a significant amount of startup code and IO that prevents the CPU from remaining active for the entire duration of the workload. Therefore, some workloads (such as scan) are able to save power by placing idle cores into low power modes. The most power hungry workload is the WordCount benchmark, which counts the occurrences of a word in a document by splitting the document, combining the partial sums of words with documents, and using the reduce tasks

to compute the total sum. WordCount is generally known as a CPU-bound workload past initialization[26], so the fact that it consumes the most power corresponds well with prior work. One interesting observation is that the so called 'sleep' workload is actually not one of the more power efficient workloads. We suspect that sleep is using an inefficient spin-loop to implement its sleeping subroutine, burning a significant amount of power during the main portion of its run-time.

Figure 10 shows a transient analysis of the WordCount benchmark. The first 20 seconds of execution are a period of low-power workload initialization, followed by a 100s primary phase where power is cyclic with Map/Reduce wave scheduling, and concluding with a trailing off of the power profile while the final Reduce straggler tasks asynchronously complete. While WordCount is the most power hungry workload, we have observed that many of the workloads show similar transient trends due to the common steps inherent in Hadoop frameworks. In terms of functional unit breakdown, the core components consume most of the power dissipated throughout execution, although core ALU power does decrease slightly during the bulk of the steady state execution.

VI. Conclusion

Researchers and chip designers must understand power dissipation in every major processor subsystem for a variety of workloads. However, the currently available tools for power measurement offer detail at too coarse a granularity to be useful for many researchers. Likewise, the best available method for detailed power estimation, cycle accurate simulation, suffers from extensive runtimes and a difficult to simulate software stack.

Towards this end, we have developed WattWatcher, a modern power estimation framework for emerging workloads. WattWatcher delivers real-time, fine-grained power estimations without the difficulty or time investment involved in simulating a deep software stack. Additionally, WattWatcher does not require significant 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 ms granularity with minimal overhead. We show that WattWatcher, when correctly calibrated, 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 form factors and manufacturing processes. Additionally, we motivate the use of WattWatcher through several real-world case studies on traditional and emerging workloads. Through the use of this methodology, it is possible to obtain a detailed power breakdown on a variety of workloads without vendor proprietary models or probes.

Acknowledgment

This work was supported in part by Semiconductor Research Corporation Task ID 2504, National Science Foundation (grants 1337393, 1218474, and 1117895), Oracle, Huawei and AMD. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views our sponsors.

References

[1] 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.

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

[3] G. Contreras and M. Martonosi, "Power prediction for intel xscale processors using performance monitoring unit events," in *ISLPED '05*, 2005.

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

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

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

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

[8] J. Dongarra *et al.*, "Energy footprint of advanced dense numerical linear algebra using tile algorithms on multicore architectures," in *CGC*, Nov 2012, pp. 274–281.

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

[10] 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.

[11] S. Wilton and N. Jouppi, "Cacti: an enhanced cache access and cycle time model," *Solid-State Circuits, IEEE Journal of*, vol. 31, no. 5, pp. 677–688, May 1996.

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

[13] G. Contreras *et al.*, "The xtrem power and performance simulator for the intel xscale core: Design and experiences," *ACM Trans. Embed. Comput. Syst.*, vol. 6, no. 1, Feb. 2007.

[14] W. Heirman *et al.*, "Power-aware multi-core simulation for early design stage hardware/software co-optimization," in *PACT*, 2012.

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

[16] "2nd Generation Intel Core Processor Family Mobile with ECC," http://www.intel.com/content/www/us/en/intelligent-systems/huron-river/2nd-gen-core-mobile-ecc-datasheet-addendum.html.

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

[18] V. M. Weaver, "Linux perf_event features and overhead," in *FastPath*, 2013.

[19] M. A. Olson, K. Bostic, and M. Seltzer, "Berkeley db," in *USENIX ATC*, 1999.

[20] T. E. Carlson, W. Heirman, and L. Eeckhout, "Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation," in *SC*, 2011.

[21] J. L. Henning, "Spec cpu2006 benchmark descriptions," *SIGARCH Comput. Archit. News*.

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

[23] Q. Xie, M. J. Dousti, and M. Pedram, "Therminator: A thermal simulator for smartphones producing accurate chip and skin temperature maps," in *ISLPED*, 2014, pp. 117–122.

[24] W. Huang *et al.*, "Hotspot: Acompact thermal modeling methodology for early-stage vlsi design," *IEEE Trans. on VLSI*, vol. 14, no. 5, pp. 501–513, May 2006.

[25] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Commun. ACM*.

[26] S. Huang *et al.*, "The hibench benchmark suite: Characterization of the mapreduce-based data analysis," in *ICDEW*, 2010.