Just-in-time Component-wise Power and Thermal Modeling

Shah Mohammad Faizur Rahman
University of Colorado at Colorado Springs
1420 Austin Bluffs Parkway
Colorado Springs, CO, USA
srahman3@uccs.edu

Qing Yi
University of Colorado at Colorado Springs
1420 Austin Bluffs Parkway
Colorado Springs, CO, USA
qyi@uccs.edu

Houman Homayoun
George Mason University
4400 University Drive
Fairfax, VA, USA
hhomayou@gmu.edu

ABSTRACT

As computer systems increasingly focus on balancing the performance and power efficiency of software applications together with temperature variations of the machine, they need to understand how software applications utilize the various architecture components differently. This paper develops a power and temperature modeling framework to provide such timely feedback, which can then be used to support a dynamic optimization system to attain better energy efficiency for applications. In particular, we present a framework that combines McPAT [17], a cycle accurate architecture simulation model, with runtime hardware performance counter statistics, to attain component-wise power consumption breakdown of applications while running at GHz speed. Our framework is able to consistently achieve 98% accuracy when compared to the actual system-level power consumption measured using a real-time power meter [1]. Finally, we present a preliminary study to demonstrate the potential of using our framework to support the optimizations of applications for better energy efficiency.

Categories and Subject Descriptors

I.6.5 [**SIMULATION AND MODELING**]: Model Development

General Terms

Performance, Measurement, Experimentation

Keywords

application categorization, machine learning

1. INTRODUCTION

As technology scaling, process variation, and thermal problems start to severely constrain both the design and utilization of Chip Multiprocessors (CMPs), computer systems need to balance the often-conflicting concerns of manageable power, temperature, and performance. To attain high performance while constrained by a power or temperature budget, these systems need to adequately reason about the management and optimization of applications, and thereby they need timely feedback of how each application dynamically utilizes the various hardware components . While many CPUs nowadays include integrated circuit level sensors to provide timely measurement of such statistics, only one sensor can typically be accommodated per core due to its physical size. Core level or processor level measurements, however, are often insufficient when reasoning about application-level power behavior. In particular, to identify potentials of improving energy efficiency through redistribution of hardware activity, an optimization system needs a a sufficiently detailed breakdown of the power trade-offs among hardware components, e.g., the distribution of dynamic power consumptions illustrated in Fig 1, which are commonly estimated through architecture level power simulations.

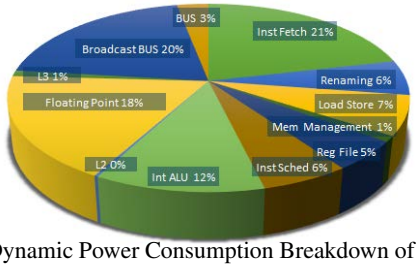


Figure 1: Dynamic Power Consumption Breakdown of Sparse Matrix Vector Multiplication code on Intel Sandy Bridge Machine estimated using our framework

Existing architecture level simulation models, e.g., CACTI [25], Wattch [5], and McPAT [17], provide detailed breakdown of the interactions between hardware components on both existing and emerging architectures. However, they are typically connected with cycle accurate simulators such as GEM5 [4] and PTLsim [28], which require applications to be compiled into a restrictive set of machine-level instructions, and an overly long time is often required to simulate a non-trivial application. On the other hand, pure statistics-based modeling approaches [22, 23, 27] use runtime hardware usage statistics to estimate the overall power consumption of microprocessors but cannot provide a sufficiently detailed breakdown of the power consumption.

This paper aims to adapt cycle-accurate simulation models for power and temperature to provide accurate timely component-wise breakdown of application behaviors while running at GHz speed. In particular, we adapted McPAT [17], an integrated framework for modeling power, area, and timing, to use runtime hardware performance counter statistics as input to compute component-wise breakdown of application-level dynamic power consumptions. The dynamic power output from McPAT is then combined with HotSpot, a thermal model [10], to estimate component-wise temperature of the microprocessor when running the applications. Such a model not only provides accurate and detailed feedback of the power ef-

CF '15, May 18-21, 2015, Ischia, Italy

http://dx.doi.org/10.1145/2742854.2742880

ficiency tradeoffs of applications in a timely fashion, it also allows future microprocessor design to better address the performance needs of existing applications.

Fig 2 illustrates the workflow of our overall framework, which includes five key components. The *Activity Estimation* component estimates McPAT activity factors, specifically runtime statistics of hardware components, from the cumulative hardware performance values collected at runtime when evaluating an application. These activity factors are then used as input to *McPAT* to estimate the dynamic power consumption of various hardware components. Then, a set of calibration formulas, generated using the *Offline calibration* component, are used to systematically correct any inaccuracies of the McPAT modeling output, by the *Result Calibration* component. Finally, the calibrated dynamic power breakdown is used as input to the *HotSpot* component, which reports component-wise temperature breakdown of the microprocessor components.

Note that while four of the components: Activity Estimation, McPAT, Result Calibration, and HotSpot, are invoked every time an input application needs to be monitored or analyzed, the *Offline Calibration* component is invoked only once when installing McPAT and HotSpot on a targeting machine. The calibration is needed to increase the accuracy level of McPAT, which has a published error rate of 22.61% [17], so that the modeling output are accurate enough to support runtime application-level optimizations. Since different applications can be mis-modeled by McPAT in dramatically different fashions, our offline calibration process uses a large number of micro benchmarks to systematically categorize application behaviors into different groups based on their runtime characteristics. A distinct calibration formula is then generated via regression analysis for each group. Once the calibration formulas are generated, they can be used by the *Result Calibration* component to quickly correct all errors from the original McPAT output. Our framework is able to achieve 98% accuracy when compared with the power consumption readings reported by a power meter [1].

The contributions of the paper include the following.

- We present a framework for accurately modeling the dynamic power consumption and temperature implications of applications in a timely fashion, by incorporating runtime performance statistics with McPAT [17] and HotSpot [10].
- We present a systematic offline calibration approach that can be used to increase the accuracy of the abstract modeling results by McPAT and other architecture simulation models.
- We present a use case study demonstrating the potential of using our modeling framework to support optimizations of applications to achieve better energy efficiency.

The rest of the paper is organized as follows. Section 2 discusses how McPAT activity factors are estimated from hardware counter values. Section 3 presents our calibration process. Section 4 describes the estimation of component-wise temperature using HotSpot. The accuracy of the power and thermal models are validated in Section 6. Sections 7 and 5 demonstrates a use case and discusses the generality of our framework. Finally, Section 8 discusses related work, and conclusions are drawn in Section 9.

2. ESTIMATING ACTIVITY FACTORS

To model the dynamic power consumed by each hardware component, McPAT [17] takes an input activity factor, which represents the statistics of utilizing the component, and then computes the overall power consumed by the component using equation (1).

$$\begin{aligned} P &= P_{dynamic} + P_{short-circuit} + P_{leakage} \\ &= \alpha \cdot C \cdot V_{dd} \cdot \delta_V \cdot f_{clk} + V_{dd} \cdot I_{sc} + V_{dd} \cdot I_{leak} \quad (1) \end{aligned}$$

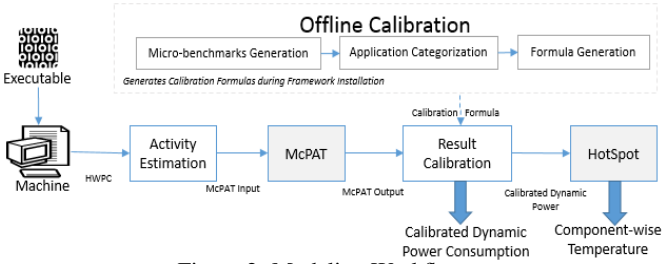


Figure 2: Modeling Workflow

Here $P_{dynamic}$ is the *dynamic power* used in charging and discharging the capacitive loads when the circuit switch states, $P_{short-circuit}$ is the power consumed by momentary short through the pull-up and pull-down devices, and $P_{leakage}$ is the power consumed due to leakage current through the transistors. $P_{dynamic}$ is calculated by multiplying the activity factor α of the component with the total load capacitance *C*, supply voltage V_{dd}, voltage swing during switching δ_V, and clock frequency f_{clk}. The activity factor α changes as applications vary their component utilization patterns.

To estimate McPAT activity factors while running the application, all the relevant hardware counters need to be monitored simultaneously. Table 1 shows our mapping between hardware counter statistics and a subset of the McPAT activity factors for an Intel Sandy Bridge microprocessor.

McPAT input	Estimation using Hardware Counters
total_cycles	*PAPI_TOT_CYC*
total_instructions	*UOPS_DISPATCHED:CORE*
int_instructions	*total_instructions-PAP_FP_INS-PAPI_BR_INS*
L2.read_misses	$\frac{L2.total\_miss * all\_loads}{(all\_stores + all_loads)}$
L3.read_accesses	*PAPI_L3_TCR*
L2.read_misses	$\frac{L3.total_miss * all\_loads}{(all\_stores + all\_loads)}$

Table 1: McPAT input estimation from hardware counters

3. CALIBRATING MCPAT RESULT

Once the activity factors of the various hardware components are estimated, McPAT uses internal models to compute the breakdown of their power consumptions. However, the output from McPAT may have a high error rate due to the following reasons.

- McPAT considers only a subset of the hardware components, specifically instruction fetch unit, renaming unit, load store unit, memory management unit, execution unit, L2 cache, L3 cache and buses. Consequently, the power consumed by applications that intensively access other components, e.g., DRAM, vector unit, may be severely underestimated.
- McPAT requires precise values for a number of power contributing factors. Since not all parameters are published by the CPU manufacturers, errors in their estimations contribute to errors in the final McPAT output.
- We estimate McPAT activity factors from hardware counter statistics. However, some activity factors, e.g., L3.read_misses, don't have a one-to-one mapping to the available counters and therefore have to be approximated, e.g., by multiplying total cache misses *PAPI_L3_TCM* with the load ratio.

Our framework aims to identify the sources of inaccuracies in the McPAT output and then correct such errors through a set of *calibration formulas*, illustrated in table 2. Since the actual errors in McPAT estimation depends on the hardware components involved and their level of intensities, different formulas are needed for applications that use the hardware components at different levels. The

following first presents our offline calibration process, which systematically generates a set of calibration formulas for different application behaviors when installing McPAT on a targeting machine. Section 3.2 then describes how to use these formulas to correct the McPAT dynamic power estimation of input applications.

3.1 Offline Calibration Formula Generation

A challenge of our offline calibration is to identify key patterns of hardware resource utilizations and then generate a distinct calibration formula for each of the identified patterns. To identify common patterns, we have generated a large set of micro-benchmarks that access the hardware components in different ways. We then identify common group behaviors from these micro-benchmarks and derive a set of criteria to effectively categorize general-purpose applications based on their hardware counter statistics. Finally, a distinct calibration formula is generated for each group via regression analysis to correct power consumption estimations by McPAT.

3.1.1 Generating Micro-benchmarks

To estimate how different activity levels of the various hardware components can impact the error rates of McPAT dynamic power output, we automatically generate a large set of micro-benchmarks, which access hardware components with different level of intensities. These micro-benchmarks cover activities in the floating point unit, vector unit, L2 and L3 cache, DRAM, among others.

Algorithm 1 Algorithm for generating micro-benchmarks

Input
$components$: targeted hardware components and the range of their intensity levels to emulate
Output
A set of generated micro-benchmarks
for each $config \in generate\_configurations(components)$ **do**
  $instructions = select\_instructions(config)$
  $block = generate\_stmt\_block(instructions)$
  $loop = generate\_loop(block)$
  $generate\_code(loop)$
end for

Algorithm 1 shows our steps for generating micro-benchmarks, by iterating over the entire configuration space entailed by the range of intensity levels for each hardware component. For each configuration, the *select_instructions* function selects a set of instructions, e.g. floating point, memory, integer, and branch instructions, to access various hardware components, by considering the latency of each instruction to ensure that the desired activity intensity of the targeted components can be achieved. Then, the *generate_stmt_block* function takes the selected instructions as input and constructs a block that schedules the selected instructions to achieve the target intensity level for each hardware component, by introducing flow dependence among the instructions that operate on the same components. Finally, the *generate_loop* function generates a loop that iterate over the block of instructions a sufficient number of times to ensure each micro-benchmark runs at least for a few seconds. Listing 1 shows a sample micro-benchmark generated using our algorithm.

Listing 1: Example micro-benchmark

```
1  void microbench() {
2    long long ii,jj;  size_t s =  sizeof(double)*AR_SZ;
3    var1 = (double*)malloc(s);  var2 = (double*)malloc(s);
4    var3 = (double*)malloc(s);  var4 = (double*)malloc(s);
5    var5 = (double*)malloc(s);  var6 = (double*)malloc(s);
6    for (ii=0; ii< REPEAT; ii++)
7      for (jj=0; jj< AR_SZ; jj++) {
8        var6[jj] = var1[jj] - var2[jj] - var3[jj]
9                  * var4[jj] * var5[jj];
10     }  }  }
```

3.1.2 Categorizing Application Behavior

After generating micro-benchmarks and collecting their hardware counter statistics by evaluating them on the target machine, our next step correlates the different behavior patterns represented by the micro-benchmarks with the actual error rates of McPAT power models. This is done by categorizing the micro-benchmarks into different groups, so that a different correcting formula can be derived for each group. In particular, for each micro-benchmark, we compare the dynamic power modeling output of McPAT with the actual dynamic power consumed by the micro-benchmark on the physical machine, measured using a Watts up? power meter [1] connected with the machine. Then, based on the correlation between hardware counter statistics and the accuracy level of McPAT modeling output, we construct a decision tree, illustrated in Fig 3, to categorize arbitrary user applications based on their intensity levels of utilizing the various hardware components.

Algorithm 2 Identify hardware performance counters

Input
hpc : hardware-counter statistics from evaluating each micro benchmark.
$thres_{HE}$: whether a hardware counter and McPAT error is strongly correlated
$thres_{HH}$: whether two hardware counters are correlated
diffs: difference between McPAT output and power meter result for each benchmark
Output
$counters$: a list of hardware counters used for categorization
Step 1:
for each hardware counter h used in McPAT modeling **do**
  $corr(h)$ = compute_correlation (hpc[h], *diffs*)
end for
Step 2: $counters = \emptyset$
for each hardware counter h in the descending order of $corr(h)$ **do**
  if $|corr(h)| > thres_{HE}$ and $\exists\, c \in McPAT\_counters$,
    $|compute\_correlation(hpc[h], hpc[c])| < thres_{HH}$ **then**
    $counters = counters \cup \{h\}$
  end if
end for
return $counters$

Algorithm 2 shows the steps we use to identify relatively independent performance counters whose activity levels have strong correlations with the accuracy of McPAT modeling results. In particular, step (1) computes the Pearson product-moment correlation coefficient (saved in variable *corr*) for each performance counter used in the McPAT modeling in relation to the overall discrepancy between McPAT modeling output and power-meter reported results. Then, step (2) of the algorithm selects independent counters that have correlation coefficients greater than $thres_{HE}$, a threshold we define to indicate whether there is a strong correlation between a counter and McPAT accuracy level. The independence of the selected performance counters are enforced by selecting counters in the descending order of their correlation coefficient with McPAT output and by selecting only counters that have no correlation ($|coefficient| \leq thres_{HH}$) with other already selected counters. We set $thres_{HE} = 0.6$ as the threshold for strong correlation and $thres_{HH} = 0.2$ as the threshold for weak or no correlation [2].

When applying this algorithm to calibrate McPAT for our Intel Sandy Bridge machine, the activity levels of four hardware counters, *DRAM accesses*, *floating point instructions*, *vector unit instructions*, and *backend stall cycles*, have been identified to strongly correlate with the accuracy of McPAT output and are independent of each other. In particular, *DRAM accesses* and *vector instructions* are not directly modeled by McPAT, so high activity levels in these components imply a greater portion of the overall dynamic power consumption being omitted in the McPAT output. The high correlation between *floating point instructions* intensity and McPAT accuracy level is a result of our approximation of McPAT floating point unit accesses using the *floating point instructions* hardware

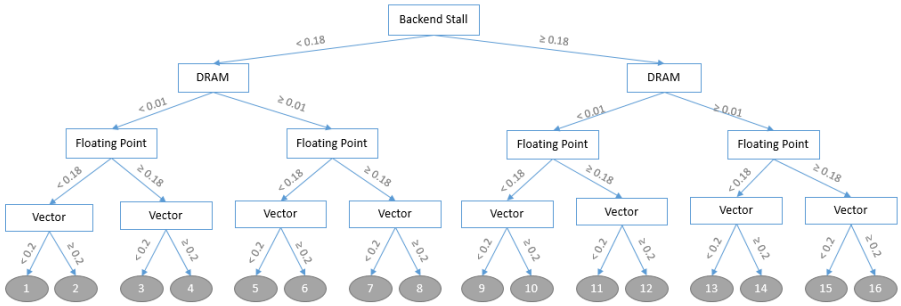


Figure 3: Decision tree for application categorization on Intel Sandy Bridge machine. Runtime statistics of the Hardware performance counters are normalized by total cycles. Each leaf represents an application behavior group.

Group	Stall	DRAM	FP	Vector	Calibration Formula
3	< 0.18	< 0.01	≥ 0.18	< 0.2	$-PAPI\_FP\_INS * 1.11 + perf :: L1 - DCACHE - LOADS * 131.43 - perf :: DTLB - LOADS * 54.89$ $- MEM\_UOPS\_RETIRED : ALL\_LOADS * 80.39 - PAPI\_BR\_CN * 3.26 + UOPS\_RETIRED : ANY * 0.66$ $+ perf :: PERF\_COUNT\_HW\_STALLED\_CYCLES\_FRONTEND * 0.56$
15	≥ 0.18	≥ 0.01	≥ 0.18	< 0.2	$-PAPI\_FP\_INS * 4.32 - OFFCORE\_RESPONSE\_0 : ANY\_RFO * 165.95 + PAPI\_L2\_TCW * 541.6$ $+ MEM\_UOPS\_RETIRED : ALL\_LOADS * 18.9 - PAPI\_TLB\_DM * 118 - -PAPI\_L3\_TCW * 318.8$ $+ PAPI\_BR\_INS * 27.3 - perf :: PERF\_COUNT\_HW\_STALLED\_CYCLES\_FRONTEND * 5.43$ $- perf :: L1 - DCACHE - LOADS * 9.14 + PERF\_COUNT\_HW\_CACHE\_L1I * 1977.2 PAPI\_VEC\_DP * 39.5$

Table 2: Calibration Formulas for application behavior groups (all hardware performance counters are normalized by total cycles). The two groups correspond to two leaves on the decision tree (Figure 3).

counter, as no existing counter can be directly used to count floating point unit accesses. Similarly, the high correlation of *backend stall cycles* is a result of our approximation of the instruction window statistics using the *backend stall cycles* performance counter.

To categorize application behavior based on the intensity levels of the selected hardware counter utilizations, we manually inspected the impact of various intensity levels on the accuracy of McPAT output to find the breaking points where the accuracy of output changes dramatically. Figures 4a, 4b, 4c and 4d show the correlation between McPAT accuracy and the intensity levels of DRAM accesses, floating point instructions, vector instructions, and backend stalls respectively. From the graphs, increases in the utilization intensity level of each hardware counter in turn results in a linear change in McPAT output accuracy until a *breaking point* which begins a line with a significantly different *slope*. The breaking points identified (marked with *red* vertical lines in the graph) for the selected hardware performance counters: *DRAM Accesses*, *Floating Point Instructions*, *Vector Instructions*, and *Backend stall cycles* are 0.01, 0.18, 0.2, and 0.18 respectively.

These breaking points are then combined together to form a decision tree, illustrated in Figure 3, which will be used to categorize an arbitrary unknown input application into one of the 16 predetermined behavior patterns from its runtime hardware statistics. Note that all the counter statistics are normalized by the total number of cycles used in application evaluation, so the categorization is not biased by the duration of the application execution. The decision tree starts with *Backend Stall Cycles*, which is at the root of the tree, followed by DRAM accesses, floating point instructions and vector instructions. The ordering of the hardware counters within the decision tree is not significant, as these counters have weak correlation among themselves.

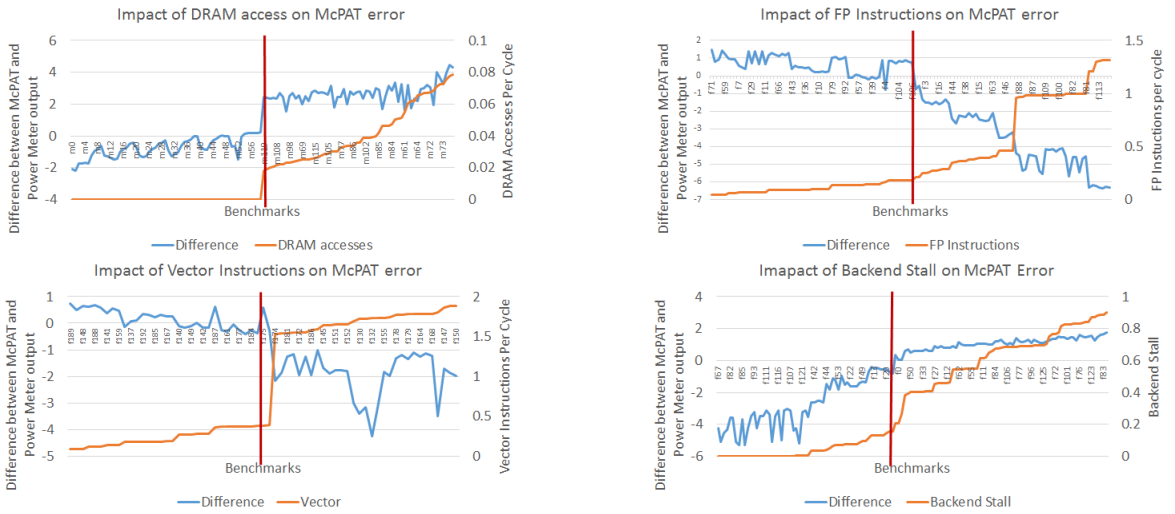
3.1.3 Generating Calibration Formulas

Once the application behavior categorization is complete, a distinct calibration formula needs to be generated to define the amount of dynamic power to be added to or subtracted from McPAT output to obtain the actual dynamic power consumed by an application. Each formula is a linear combination of the hardware counter statistics that have strong correlation with the difference between McPAT output and the actual power consumption when categorizing our auto-generated micro-benchmarks.

Table 2 illustrates two of the formulas we generated for the Intel Sandy Bridge machine for two application groups, 3 and 15, from the decision tree in Figure 3. Each formula is generated through two steps: (1) identifying hardware counters whose runtime statistics are not modeled accurately by McPAT, and (2) identifying the actual ratios of the hardware counter statistics that are mis-modeled by McPAT and then using the ratios as coefficients of the corresponding term in the final calibration formula. The hardware counters are selected by computing the Pearson product-moment correlation coefficient of each hardware counter statistics in relation to the difference between McPAT dynamic power estimation and power meter output, and then choosing the counters that have strong correlation (> 0.6, Salkin [2]) with the difference. The coefficients of the counters are calculated by applying multivariate Ordinary Least Square (OLS), which is a standard method for identifying unknown parameters in a linear regression model.

Our approach essentially uses regression analysis to generate a linear combination of the hardware counter statistics to predict the difference between McPAT dynamic power estimation and power meter output. Therefore, the correctness of these calibration formulas depends on the validity of the following two hypotheses:

- *For each application, a linear combination of its hardware counter statistics can be used to represent the difference between McPAT dynamic power estimation and power meter output.* This difference can come from two sources: i) the hardware components mis-modeled by our adapted McPAT, the amount of which can be estimated by multiplying activity factors of each hardware component with their actual ratios of mis-modeling, and ii) the hardware components currently not being modeled by McPAT, the amount of which can be estimated by multiplying activity factors of each hardware component with the cost of performing the activities. Since the contribution of both sources can be expressed as a linear combination of hardware performance counter statistics, their sum can be represented in a similar fashion.



(c) Vector Instructions (d) Backend Stalls

Figure 4: Estimating breaking points for selected hardware counters

- *The decision tree constructed by our framework correctly categorizes applications into groups, so that for each group, a formula can be used to accurately calculate the difference between McPAT dynamic power estimation and power meter output.* Our application categorization algorithm identifies behavior patterns that utilize hardware resources with similar levels of intensity, so each group would be mis-modeled by McPAT in a similar fashion. To estimate the accuracy of our regression analysis, we verified their coefficients of determination (R^2), where a coefficient that is close to one indicates the model is accurate, We found $R^2 > 0.9$ for all the formulas we generated on the Intel Sandy Bridge machine, indicating the regression analysis is accurate.

3.2 Applying the Calibration Formulas

After generating the set of calibration formulas through *Offline Calibration*, these formulas are used by the *Result Calibration* component in Figure 2 to calculate the amount of dynamic power that need to be added to or subtracted from McPAT modeling output for an arbitrary input application. In particular, the *Result Calibration Component* takes two inputs: i) McPAT power modeling result, and ii) runtime hardware counter statistics collected when evaluating the application. It then uses the pre-generated decision tree and calibration formulas to modify the McPAT modeling output and produces a calibrated dynamic power estimation through the following three steps.

1. Categorize the input application as a member of one of the pre-generated behavior pattern groups, by navigating a decision tree (Figure 3) pre-generated by the *offline calibration* process, using the runtime hardware performance counter statistics collected when evaluating the application.
2. Find the calibration formula pre-customized for the selected group and invoke it using the runtime counter statistics of evaluating the application as input, to compute the amount of McPAT power estimation to be adjusted.
3. The McPAT modeling output is modified accordingly, and the calibrated dynamic power consumption is output.

While the result calibration component is invoked every time an application is modeled by our framework, it takes minimal time as it uses pre-generated decision trees and formulas.

4. ESTIMATING TEMPERATURE

As shown in Figure 2, after producing the calibrated power consumption breakdown of an application, our framework uses the *HotSpot* thermal model [10] to estimate the temperature of each hardware component. While modern microprocessors (e.g. Intel Sandy Bridge) are now equipped with circuit-level temperature sensors, only one sensor can be placed per core, limiting them from measuring the temperature of individual hot spots within the processor, an objective targeted by our framework.

To estimate the temperature of a microprocessor, HotSpot [10] requires a floor plan describing the size and location of all the components inside the processor, and the power consumption of each component in the floor plan. Using these as input HotSpot models temperature in two phases. First, it calculates the steady state temperature, which represents the initial temperature of the processor component. Then, for each power trace, it generates the transient temperature of all the available hardware components defined in the floor plan. This HotSpot output can be used to improve the accuracy of McPAT leakage power estimation, which depends on the temperature of the components being modeled, in addition to supporting temperature-aware optimization of application executions.

5. GENERALITY AND EFFECTIVENESS

Our work enables architectural power and thermal models to guide application optimizations by providing timely feedback of detailed power consumption and temperature breakdown on real hardware. We have demonstrated the effectiveness of our approach using McPAT [17] and HotSpot [10] to model the dynamic power consumption and temperature respectively on an Intel Sandy Bridge machine. While our framework currently works with only McPAT and HotSpot, our approach can be similarly used to adapt other architecture power or thermal models to use hardware counter statistics as input and to produce accurate timely feedback to guide application level optimizations. To extend our work for a different power/temperature model or a different machine, the mapping between the model specific hardware activities and and the machine-specific hardware counter statistics need to be modified. Further, the offline calibration component, shown in Figure 2, needs to be invoked to generate a different decision tree for application categorization and a different set of calibration formulas. Our current calibration pro-

cess requires manual selection of the hardware counters and identification of the breaking points of their runtime statistics. However, machine learning techniques can be used to automate such procedures, which is a subject of our future work.

6. EXPERIMENTAL RESULTS

To validate the accuracy of our calibrated McPAT model, we compare its modeling output for a variety of computational kernels and benchmark applications, with the actual dynamic power consumption of the physical machine when running these applications, measured using a Watts up? Pro power meter [1][1]. Similarly, the temperature output of our adapted HotSpot thermal model is compared with the actual temperature of the processor measured using an on-chip hardware thermal sensor. The following first introduces the benchmarks and machines we used to validate our calibrated models. The results are then discussed in detail.

6.1 Benchmarks and Machine Configuration

We selected seven scientific kernels, including three stencil computations, (jacobi7, jacobi27, and gauss7), three dense matrix computations (gemm, gemv, and ger), and one sparse matrix computation (SPMV); a library benchmark (LINPACK [7]); two synthetic benchmarks (WhetStone and DhryStone); and 10 application benchmarks from NAS [3]. The selection aims to represent a wide variety of application behaviors in the scientific computing domain, which are traditionally considered the main target of high performance computing but have been increasingly put under the power and temperature constraints by modern architecture design.

Each benchmark is first compiled using gcc 4.7 with the *-O3* optimization flag and then evaluated on a 12-core Intel Xeon E5-2420 Sandy Bridge-EN 1.9GHz Processor. The *psrun* utility from PerfSuite [16], which uses PAPI[18] underneath, is used to collect the runtime statistics of each benchmark evaluation. The hardware counter statistics are then used as input to our calibrated McPAT model, and the McPAT modeling output is compared with the actual dynamic power consumption of the whole machine measured using a power meter [1]. When evaluating each benchmark, the machine is kept idle otherwise, so the dynamic power of the whole machine equals that from evaluating the benchmark.

We used the on-chip physical thermal sensors inside the Intel Sandy Bridge processor to validate our thermal modeling results. Since these sensors can only output temperature of the whole core, we validated our thermal modeling results only at the core level.

6.2 Calibrated Power Model Validation

For each benchmark, Figure 5 compares its dynamic power consumption estimated by our calibrated McPAT model with that measured by the power meter and that reported by McPAT without any calibration. From the graph, the dynamic power output by our calibrated McPAT model closely matches that reported by the power meter, with the average error rate about 2% for all the benchmarks. Note that the power meter itself has a 1.5% measurement error rate, which roughly matches those of our calibrated power model.

When comparing the power meter output with the power modeling output of McPAT before applying the calibration formulas, we see that our calibration formulas have successfully corrected most of the mis-predictions by the out-of-the-box McPAT. In particular, without calibration, the average error rate of the McPAT modeling output is about 25.1%, which roughly matches the error rates reported by [17] when using McPAT to model several other processors. Typically, McPAT underestimates the overall system dynamic power consumption if a benchmark (e.g., ger, jacobi7, mg, cg, and ua) intensively uses some components (e.g., DRAM) that are not directly modeled by McPAT. On the other hand, it may overestimate the actual power consumed when the input hardware counter statistics do not fully represent the activity intensities of the hardware components. Our calibration process combines application categorization and regression analysis to fully accommodate the various scenarios, thereby able to reduce the average error rate to 2%.

[1]We also collected RAPL coutners present in Intel Sandy Bridge machine. However, as the experiment did not finish in time, we have not used it to validate our model

6.3 Temperature Model Validation

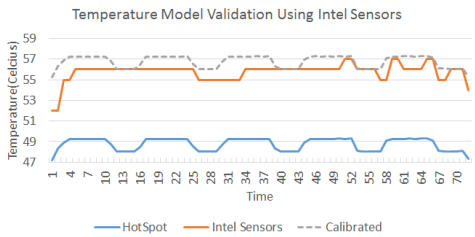


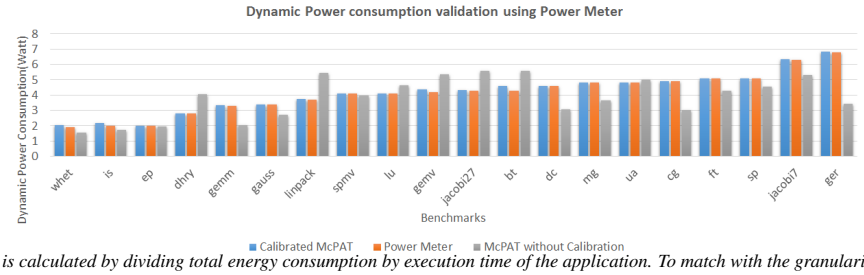
Figure 6: Thermal Model Validation

To determine whether our HotSpot modeling output can accurately capture the instantaneous temperature variations caused by changes of application behaviors, we have modified the *jacobi7* stencil kernel to alternate between two stages: (1) performing a fixed number of iterations of stencil computations and (2) sleeping for a few seconds. Figure 6 compares the core-level temperature variations estimated by HotSpot with those measured by using Intel on-chip hardware thermal sensor, throughout the duration of evaluating the modified *jacobi7* code. From the graph, the two alternating stages are clearly reflected by our HotSpot temperature output, as by the temperature variations measured by the on-chip thermal sensor. While the HotSpot modeling output is consistently below the temperature measured by the sensor, their alternating pattern closely resemble each other. After adding a constant ($8\,^{\circ}$C) to the HotSpot output as part of the calibration process, it achieves an error rate of 1.8%, which is within the reported error rate of the temperature sensors.

7. GUIDING OPTIMIZATION DECISIONS

The feedback provided by our modeling framework can be used to better balance the power consumptions of different hardware components when executing the software applications. For example, some program transformations, e.g., loop unrolling, can be used to boost instruction level parallelism and thereby change the relative activity levels of the processor front-end (e.g., Instruction Fetch Unit) and execution units (e.g., Floating Point Unit). Applying such transformations in a way that can adjust the relative balance of activity levels of hardware components can thereby reduce wasted energy in overly active components.

To demonstrate the viability of such an optimization approach, we have applied loop unrolling with a wide variety of different unrolling factors to the innermost loops of a set of matrix (*gemm, gemv, ger*), stencil (*jacobi7, jacobi27, gauss*), and BLAS kernels (*daxpy, dscal*).Then, the relative balance of the dynamic power consumed by the Instruction Fetch Unit (IFU) and Floating Point Unit (FPU) are plotted and correlated with the overall energy efficiency of each kernel evaluation, represented by the energy-delay product (EDP) of each evaluation [9] (the lower the EDP is, the more energy efficient is the optimized code).



**Dynamic power consumption is calculated by dividing total energy consumption by execution time of the application. To match with the granularity of Watts up? Pro power meter, we ran McPAT at per second granularity. Each benchmark was evaluated at least for ten seconds to reduce noise.*

Figure 5: Validating the Calibrated Model using Power Meter

Figure 7 shows variations of the relative balance between the power consumed by IFU and FPU of the benchmarks when different factors are used to unroll the innermost loop of the kernel, together with the overall energy efficiency (EDP) of each differently unrolled kernel. From the graph, a strong correlation (> 0.96) can be observed between the IFU/FPU ratio and the EDP (overall energy efficiency) for all the kernels, where a reduction in IFU/FPU ratio, in turn reduces EDP and improves the overall energy efficiency of the code. Therefore, the component-wise breakdown of power consumption reported by our framework can be used to guide loop unrolling by reducing the IFU/FPU ratio and thereby improving the overall energy efficiency of applications.

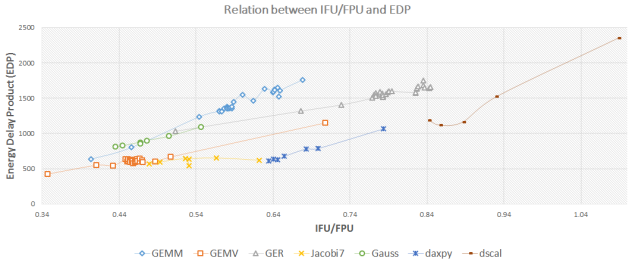


Figure 7: Relation between power consumption ratio of IFU/FPU and Energy Delay Product

8. RELATED WORK

Most of existing work [25, 5, 17, 10] on modeling the power and temperature of microprocessors is based on architecture level simulation and designed for offline use due to the prohibitively long time required by instruction-level simulation. Our framework adapts such models to provide accurate and detailed dynamic power feedback of hardware components by using hardware counter statistics as input, to make them order of magnitude faster.

Existing work has correlated performance counters with the power consumptions of computer systems both through regression-based approach [6, 22, 8, 26] and by modeling the activity intensities of the function units within the processors [23, 14, 12, 19, 13, 11, 27]. In particular, our work is similar to that by Joesph et al [14], who used performance counters with the Wattch [5] and SimpleScalar simulators to model component-wise power consumption. Our work is unique in that we focus on developing a systematic calibration approach to improve the accuracy of such models so that they can be used to guide application-level dynamic optimizations.

We used McPAT [17] and HotSpot [10] to model the component-wise power consumption and temperature of microprocessors. Other existing power and temperature modeling frameworks include CACTI [25], which estimates dynamic and leakage power by modeling devices based on the industry standard ITRS roadmap, and Wattch [5], which computes dynamic power dissipation from switching events obtained from architecture simulation and compute capacitance models of the micro-architecture components. Our framework can be extended to use other power/temperature models. Our contribution is to extend these modeling frameworks to use hardware counter statistics as input and to demonstrate a regression based approach that can significantly enhance the accuracy of the modeling results so that they can be used to guide application-level optimizations by developers or compilers.

Valluri and John [24] studied the impact of different compiler optimization levels on processor power and energy consumption on a Dec Alpha 21064 CPU. Kandemir et al [15] studied the power/energy impact of both low-level compiler optimizations and three loop reordering optimizations on both the CPU and the memory system using a matrix multiplication computation. Seng and Tullsen [21] studied the power/energy effects of different compiler optimization levels and three compiler optimizations: loop unrolling, vectorization, and function inlining, on an Intel Pentium 4 Processor. Previous research has shown that compiler optimizations [15] have high impact on system power consumption, and that optimizing for performance is not the same as optimizing for power consumption [24]. Rahman et al [20] used a hardware-counter-based system-level power model to guide compiler optimizations. Our framework serves a similar purpose and aims to provide timely accurate component-wise power and temperature feedback to guide optimizations in a more meaningful fashion.

9. CONCLUSION

This paper presents a framework that uses existing architectural models to provide accurate timely feedback about the power consumption and temperatures of hardware components, thereby enabling software applications to be better optimized for energy efficiency by making quick informed decisions. Our contributions include adapting existing architectural models to use hardware counter statistics as input and a systematic approach to calibrate the modeling output to produce accurate results that match those measured by physical power meters or hardware sensors. The dynamic power consumption reported by our framework is able to attain 98% accuracy when compared with those actually measured using a Watts up? Pro power meter [1]. Our future work will use machine learning techniques to automate the calibration process and to use the results to guide more sophisticated runtime optimization and scheduling decisions to enhance overall system energy efficiency.

10. ACKNOWLEDGEMENTS

This research is supported by National Science Foundation of USA under grant CCF-1261811, CCF-1261778, and CCF-1421443.

11. REFERENCES

[1] *Watts up? Pro (http://www.wattsupmeters.com)*.

[2] *Encyclopedia of measurement and Statistics*. Sage Publications Ltd,, London :, 2007. Includes index and bibliographical reference.

[3] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, H. D. Simon, V. Venkatakrishnan, and S. K. Weeratunga. The NAS parallel benchmarks. Technical report, The International Journal of Supercomputer Applications, 1991.

[4] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood. The gem5 simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7, Aug. 2011.

[5] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: a framework for architectural-level power analysis and optimizations. In *Proceedings of the 27th annual international symposium on Computer architecture*, ISCA '00, pages 83–94, New York, NY, USA, 2000. ACM.

[6] G. Contreras and M. Martonosi. Power prediction for Intel XScale processors using performance monitoring unit events. In *Proceedings of the 2005 international symposium on Low power electronics and design*, pages 221–226, New York, NY, USA, 2005. ACM.

[7] J. Dongarra, P. Luszczek, and A. Petitet. The LINPACK benchmark: past, present and future. *Concurrency and Computation: Practice and Experience*, 15(9):803, 2003.

[8] B. Goel, S. A. McKee, R. Gioiosa, K. Singh, M. Bhadauria, and M. Cesati. Portable, scalable, per-core power estimation for intelligent resource management. In *Green Computing Conference*, pages 135–146. IEEE, 2010.

[9] R. Gonzales and M. Horowitz. Energy dissipation in general purpose processors. *IEEE Journal of Solid State Circuits*, 31:1277–1284, 1995.

[10] W. Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron, and M. R. Stan. Hotspot: A compact thermal modeling methodology for early-stage VLSI design. *IEEE Trans. Very Large Scale Integr. Syst.*, 14(5):501, May 2006.

[11] W. Huang, C. Lefurgy, W. Kuk, A. Buyuktosunoglu, M. Floyd, K. Rajamani, M. Allen-Ware, and B. Brock. Accurate fine-grained processor power proxies. In *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-45, pages 224–234, Washington, DC, USA, 2012. IEEE Computer Society.

[12] C. Isci, G. Contreras, and M. Martonosi. Live, runtime phase monitoring and prediction on real systems with application to dynamic power management. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 39, pages 359–370, Washington, DC, USA, 2006. IEEE Computer Society.

[13] H. M. Jacobson, A. Buyuktosunoglu, P. Bose, E. Acar, and R. J. Eickemeyer. Abstraction and microarchitecture scaling in early-stage power modeling. In *HPCA*, pages 394–405. IEEE Computer Society, 2011.

[14] R. Joseph and M. Martonosi. Run-time power estimation in high performance microprocessors. In *Proceedings of the 2001 international symposium on Low power electronics and design*, pages 135–140, New York, NY, USA, 2001. ACM.

[15] M. Kandemir, N. Vijaykrishnan, and M. J. Irwin. Power aware computing. chapter Compiler optimizations for low power systems, pages 191–210. Kluwer Academic Publishers, Norwell, MA, USA, 2002.

[16] R. Kufrin. Perfsuite: An accessible, open source performance analysis environment for linux. In *In Proc. of the Linux Cluster Conference, Chapel*, 2005.

[17] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi. Mcpat: an integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 42, pages 469–480, New York, NY, USA, 2009. ACM.

[18] P. J. Mucci, S. Browne, C. Deane, and G. Ho. Papi: A portable interface to hardware performance counters. In *In Proceedings of the Department of Defense HPCMP Users Group Conference*, pages 7–10, 1999.

[19] M. D. Powell, A. Biswas, J. S. Emer, S. S. Mukherjee, B. R. Sheikh, and S. M. Yardi. Camp: A technique to estimate per-structure power at run-time using a few simple parameters. In *HPCA*, pages 289–300. IEEE Computer Society, 2009.

[20] S. F. Rahman, J. Guo, and Q. Yi. Automated empirical tuning of scientific codes for performance and power consumption. In *Proceedings of the 6th International Conference on High Performance and Embedded Architectures and Compilers*, HiPEAC '11, pages 107–116, New York, NY, USA, 2011. ACM.

[21] J. Seng and D. Tullsen. Exploring the potential of architecture-level power optimizations. In B. Falsafi and T. VijayKumar, editors, *Power-Aware Computer Systems*, volume 3164 of *Lecture Notes in Computer Science*, pages 132–147. Springer Berlin Heidelberg, 2005.

[22] K. Singh, M. Bhadauria, and S. A. McKee. Real time power estimation and thread scheduling via performance counters. *SIGARCH Comput. Archit. News*, 37(2):46–55, July 2009.

[23] V. Spiliopoulos, A. Sembrant, and S. Kaxiras. Power-sleuth: A tool for investigating your program's power behavior. In *Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS), 2012 IEEE 20th International Symposium on*, pages 241–250, Aug 2012.

[24] M. Valluri and L. John. Is compiling for performance == compiling for power? In *Interaction between Compilers and Computer Architectures*, volume 613 of *The Springer International Series in Engineering and Computer Science*, pages 101–115. Springer US, 2001.

[25] S. J. E. Wilton and N. P. Jouppi. Cacti: An enhanced cache access and cycle time model. *IEEE Journal of Solid-State Circuits*, 31:677–688, 1996.

[26] Y. Xiao, R. Bhaumik, Z. Yang, M. Siekkinen, P. Savolainen, and A. Yla-Jaaski. A system-level model for runtime power estimation on mobile devices. In *Proceedings of the 2010 IEEE/ACM Int'L Conference on Green Computing and Communications & Int'L Conference on Cyber, Physical and Social Computing*, pages 27–34, Washington, DC, USA, 2010. IEEE Computer Society.

[27] C. Yoon, D. Kim, W. Jung, C. Kang, and H. Cha. Appscope: Application energy metering framework for android smartphones using kernel activity monitoring. In *Proceedings of the 2012 USENIX Conference on Annual Technical Conference*, USENIX ATC'12, pages 36–36, Berkeley, CA, USA, 2012. USENIX Association.

[28] M. T. Yourst. Ptlsim: A cycle accurate full system x86-64 microarchitectural simulator. In *ISPASS*, 2007.