# Fine-Grain Power Breakdown of Modern Out-of-Order Cores and Its Implications on Skylake-Based Systems

**Article** *in* ACM Transactions on Architecture and Code Optimization · December 2016

DOI: 10.1145/3018112

CITATIONS
37

READS
559

**4 authors**, including:

Jawad Haj-Yahya
ETH Zurich
**44** PUBLICATIONS **209** CITATIONS

SEE PROFILE

Ahmad Yasin
Intel
**18** PUBLICATIONS **405** CITATIONS

SEE PROFILE

Avi Mendelson
Technion - Israel Institute of Technology
**166** PUBLICATIONS **2,287** CITATIONS

SEE PROFILE

# Fine-Grain Power Breakdown of Modern Out-of-Order Cores and Its Implications on Skylake-Based Systems

JAWAD HAJ-YIHIA, University of Haifa
AHMAD YASIN, Intel Corporation
YOSI BEN ASHER, University of Haifa
AVI MENDELSON, Technion – Israeli Institute of Technology

A detailed analysis of power consumption at low system levels becomes important as a means for reducing the overall power consumption of a system and its thermal hot spots. This work presents a new power estimation method that allows understanding the power breakdown of an application when running on modern processor architecture such as the newly released Intel Skylake processor. This work also provides a detailed power and performance characterization report for the SPEC CPU2006 benchmarks, analysis of the data using side-by-side power and performance breakdowns, as well as few interesting case studies.

CCS Concepts: • **Computing methodologies → Modeling and simulation**; • **Hardware → Power estimation and optimization**

Additional Key Words and Phrases: Power, energy, performance-counters

**ACM Reference Format:**
Jawad Haj-Yihia, Ahmad Yasin, Yosi Ben Asher, and Avi Mendelson. 2016. Fine-grain power breakdown of modern out-of-order cores and its implications on skylake-based systems. ACM Trans. Archit. Code Optim. 13, 4, Article 56 (December 2016), 25 pages.
DOI: http://dx.doi.org/10.1145/3018112



## 1. INTRODUCTION

*Motivation & Background*. Power consumption has become a critical issue that affects the further development of processors. High power consumption adds challenges to the system design due to the effects of high thermal output, high current requirements, and battery life and electricity costs. In order to raise the awareness, Green500 [Top 500 2013; The Green500 2013] lists the most energy-efficient supercomputers twice a year according to the ratio of performance and power consumption (as opposed to pure performance ranking). Furthermore, quantifying the power consumption of individual applications is a critical component for software-based power capping [Haj-Yihia et al. 2015], scheduling, and provisioning techniques in modern datacenters. Fine-grain power breakdown is essential for other fronts like power and performance optimization guidelines.

This work is supported in part by the European Research Council Advanced Grant DAL No 267175 and the National Science Foundation (CNS-1117280, CCF-1218323, CNS-1302260, CCF-1438992, and CCF-1533885). The views expressed herein are not necessarily those of the NSF. Professor Wood has significant financial interests in AMD, Google, and Panasas.
Authors' addresses: J. Haj-Yihia, A. Yasin, and Y. B. Asher, Haifa University, 199 Aba Khoushy Ave, Mount Carmel, Haifa 3498838, Israel; emails: jhajyihi@campus.haifa.ac.il, ahmad.yasin@intel.com, yosi@cs.haifa.ac.il; A. Mendelson, Technion – Israeli Institute of Technology, Technion City, Haifa 3200003, Israel; email: avi.mendelson@tce.technion.ac.il.

DOI: http://dx.doi.org/10.1145/3018112

Given the importance of power, it is critical for system designers and software developers to be able to observe, estimate and analyze power consumption. Power estimation is fundamental for optimal scheduling in operating systems and for the development of power-efficient mechanisms in processor firmware. Therefore, it has recently attracted many researchers and yielded different important reference guides. Prior research indicates that power consumption can be estimated via monitoring performance counters, such as cycles, instructions or cache accesses. These works proposed how to model power at coarse levels such as processor package or processor core mostly in simulations and some in real systems too. They can be categorized in top-down or bottom-up approaches. We summarize these in the Related Works section and provide a table with high-level comparison.

*Problem & Scope*. So far, most of the software optimization guidelines do not pay extra attention to energy consumption and often assume that optimizing software for performance is the same as optimizing the code for energy [Intel 2014; Isci et al. 2006]. This assumption is not always true. For example, in Rotem et al. [2014] the authors compare between the "Race to Halt" schedule – that suggests running the system at maximum speed to be able to put the system in a sleep state for the longest time possible – and the Lowest-Frequency-Mode mechanism that calls to run the system at the lowest frequency possible so that the active power will be minimized. This article concludes that different systems may have different optimization points to minimize energy consumption. Such an energy optimization point mainly depends on the ratios between the system power, leakage power, and the dynamic power the system consumes during its execution time. Finding the right mechanism that could bring the system to the best energy optimization point, usually involves a deep understanding of the power breakdown of the different micro-architectural components of the system.

For example, suppose we decide to impose the race-to-halt policy, the system needs to take care that no single execution FUB (Functional Unit Block) will cause a thermal hot spot that would reduce the frequency of the entire system. In this case, distributing the work among different execution units, e.g., Floating-Point (FP) scalar operations and SIMD/vector operations will reduce the thermal hot-spot but may also cause increase of the overall energy due to extra time the system is now active. Thus, the optimal operational point for energy is not clear in such a case and we need new tools and power breakdown of components in order to determine what optimizations to embrace for each scenario.

The scope of this work focuses on CPU-bound workloads and on single-threaded core processors. CPU bound implies the workload does not include much CPU idleness (C-states [Intel 2014]). In addition, our tool assumes that the CPU performance point (P-states [Intel 2014]) and temperature do not change in the middle of the benchmarks runs.

*Our Approach*. We have developed a tool that accurately breaks down the power consumption of modern processors at sub-processor core granularity on real systems.

Our work reports the actual power (in Watts) of the Intel's state of the art core microarchitecture at abstracted domains: Front-end portion (responsible for instructions/fetch operations), execution and out-of-order engine and Memory subsystem. We further model power consumption at processor core-level and manage to achieve accuracy levels of package-level energy counters provided by the hardware vendors.

*Contributions*. In this article, we make the following contributions:

—We describe a methodology for modeling per-domain power consumption within out-of-order cores of modern (real) systems (Section 3) and provide a high-level comparison of difference among prior methods (survey) (Related Works).

—The method relies on a select list of performance counters that were chosen for an empirical reason. We also identify how the specific performance counters correlate to the power consumption of each sub-domain.
—The tool shows up to 7% error rate relative to Intel RAPL [David et al. 2010] power reporting
—We demonstrate/validate the method on Intel's 6th generation Core (codename Skylake) using designated micro-benchmarks (Section 5.2) where we study their power and performance breakdowns and how it varies with common software optimizations such as loop unrolling or vectorization.
—We characterize power consumption of the SPEC CPU2006 benchmarks on Skylake (Sections 5.3 and 5.4) study few interesting cases (Section 5.5), and include detailed power-performance report (Appendix).
—We developed a tool that models the power consumption of Intel's Core™ and tuned it for the Skylake microarchitecture. The tool presents overall core power, per-domain breakdown and the ability for over-time estimation (Section 4). We make this tool available to the research community as an open-source [Haj-Yihia et al. 2016].

## 2. BACKGROUND

This section provides necessary power, microarchitecture and performance analysis background used later by the Methodology and Results sections.

### 2.1. Power

Power is the rate at which energy is consumed; the unit of power is watts (W), which is Joules per second. Power consumption in CMOS circuits can be described by a dynamic and a static part. The static part is mainly due to leakage, and the dynamic part can be further divided into switching power and short-circuit power. Switching power has the largest share of power consumption in CMOS circuits when the circuit is active, switching power can be expressed as shown in Equation (1).

*Equation* (1): *Power consumption in CMOS circuits*

$$P = P_{dyn} + P_{sta} = P_{swi} + P_{sc} + P_{leak}$$
$$P_{swi} = \alpha C_L V^2 f$$

where V is the supply voltage, $C_L$ are load and parasitic capacitances that are charged and discharged with frequency $f$ when the corresponding component is active [Bhunia and Mukhopadhyay 2010]. The switching activity $\alpha$ is highly dependent on the applications running and their input data.

### 2.2. Processor Energy Measurements

Intel introduced the Running Average Power Limit (RAPL [David et al. 2010]) feature with the Sandy Bridge microarchitecture [Rotem et al. 2012]. RAPL reports the energy consumption of the system components listed in Table I.

These available counters enable reading the energy consumption of the package, cores, graphics (at client processor) and memory domains (at server processor). In Skylake, a new capability was added to read the whole platform energy consumption (energy consumed from the battery or power source). RAPL data can be configured and examined by reading Model-Specific Registers (MSRs) that require privileged kernel mode access today.

The PP0_ENERGY counter accounts for all cores within a multicore processor combined with the Last-Level-Cache (LLC) to which these cores are connected. It is the closest publicly available reference to the granularities our work models.

To figure out the *average-power* consumed by some benchmark at one of the processors domains, assuming no other programs are running except our benchmark, then

Table I. List of Available Energy Telemetries for Intel Processors

| Domain | Description |
|---|---|
| PKG_ENERGY | Energy of the whole processor package |
| PP0_ENERGY | Energy of Power Plane 0 domain (Processor IA Cores domain and Last-Level-Cache - LLC) |
| PP1_ENERGY | Energy of Power Plane 1 domain (specific to client processors where it is the integrated Processor Graphics) |
| DRAM_ENERGY | Energy of the DRAM domain (specific to server processors) |
| PLATFORM_ENERGY | Energy of the entire platform including IA Core, Integrated Graphics and the System Agent and other platform components. (new in Skylake) |

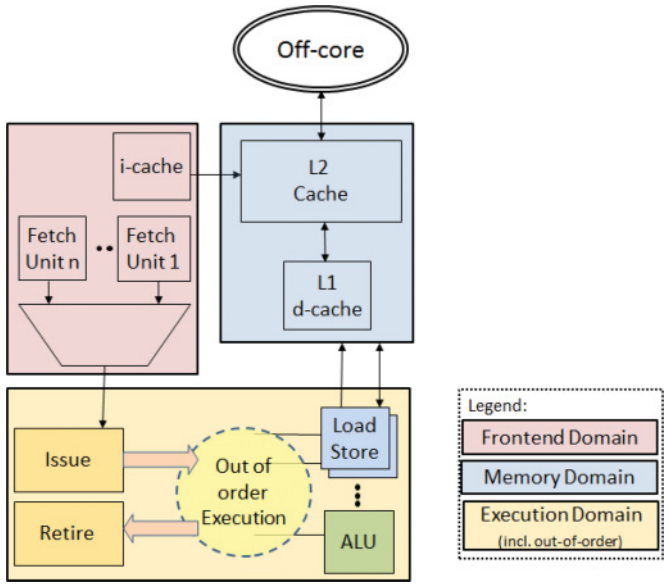


Fig. 1. Abstraction of a modern processor core microarchitecture.

we can sample the relevant RAPL counter (from Table I) before and after executing the benchmark; the difference between the two samples gives the energy consumed by the benchmark at the relevant domain, dividing the energy by the execution time gives us the average-power consumed by the benchmarks as $Energy = average\text{-}power \cdot execution\_time$.

### 2.3. Core Microarchitecture

At a high level, the pipeline of a modern out-of-order processor can be divided into two main portions: Frontend and a Backend. The Frontend is responsible for fetching instructions from memory and translating them into micro-operations (uops). These uops are fed to the Backend. The Backend is responsible for scheduling, executing and retiring (commit) these uops per the original program's order. The Backend can be further divided into Execution and Memory-domains. The Execution domain includes the out-of-order scheduler and the execution units while the Memory domain includes memory subsystem portions, private to the core including the data cache and second level cache, as depicted in Figure 1.

### 2.4. Counter-Based Power Estimation

Recently many studies have been carried out on run-time power monitoring using performance monitoring counters (PMCs). PMCs are available in various processors for performance measurement, but have been successfully used for power estimation. Bellosa [2000] discovered a strong correlation between system performance events, such as floating-point operations or cache misses, and required energy. The dynamic power consumption of a system with $n$ PMCs can then be described by multiplying the performance counter readings $PMC_i$ with the corresponding energy per performance event $e_i$ divided by the sampling interval. Previous works, such as Powell et al. [2009], Isci et al. [2006] and Bertran et al. [2010, 2013a] estimate the power of core sub-units by using a few counters, the work was done for a relatively old processor simulator (not real system) and their scheme does not report actual power or energy but pseudo-Joule metric.

### 2.5. Performance Analysis

We used the Top-down Microarchitecture Analysis Method (TMAM) [Yasin 2014; Intel 2014] for performance analysis. TMAM simplifies cycle-accounting (the process of identifying costs of performance bottlenecks, also often called CPI breakdown) for out-of-order cores using microarchitecture-independent metrics organized in one simple hierarchy.

At the top level, TMAM employs a single point of division where issue-pipeline slots are divided into four main categories: Frontend Bound, Backend Bound, Bad Speculation, and Retiring. The latter two denote non-stalled slots while the former two denote stalls. A simple decision tree is used: if a slot is utilized by some operation, it would be classified to Retiring or Bad Speculation, depending on whether it eventually gets retired. Unutilized slots are classified to Backend Bound if the back-end portion of the pipeline is unable to accept more operations (also known as Backend-stall [Yasin 2014]), or Frontend Bound where no operations are delivered while there was no Backend-stall.

The method further divides Backend Bound into Memory Bound or Core Bound in the second hierarchy level, depending on whether the Backend-stalls are due to memory or non-memory operations, respectively. In the results section, we show Memory Bound and Core Bound instead of Backend Bound alongside the other three top-level categories. This is possible with the Skylake version of the TMAM method where first and second hierarchy levels adopt the same counting domain in TMAM_Metrics electronic files posted at Intel [2015]. This representation is chosen in order to ease the comparison with our own power breakdown model.

## 3. POWER BREAKDOWN METHODOLOGY

For modeling the power of the CPU core and its sub-domains, we have used a linear function of performance counters while, for each performance counter, we have associated a weight as shown in Equation (2).

*Equation (2). Core and sub-domains power estimation using performance counters*

$$Power = Leakage + \sum_{i=0}^{n} \frac{\text{PerfCounter}_i}{time}.\text{Weight}_i$$

To determine the values of the weights, we used a set of representative micro-benchmarks that stress and train each part of the core domains. These micro-benchmarks were run both on the simulator and a real platform. By running a micro-benchmark on the simulator, we got the power for the core and its sub-domain for the specific micro-benchmark. And by running the micro-benchmark on the real

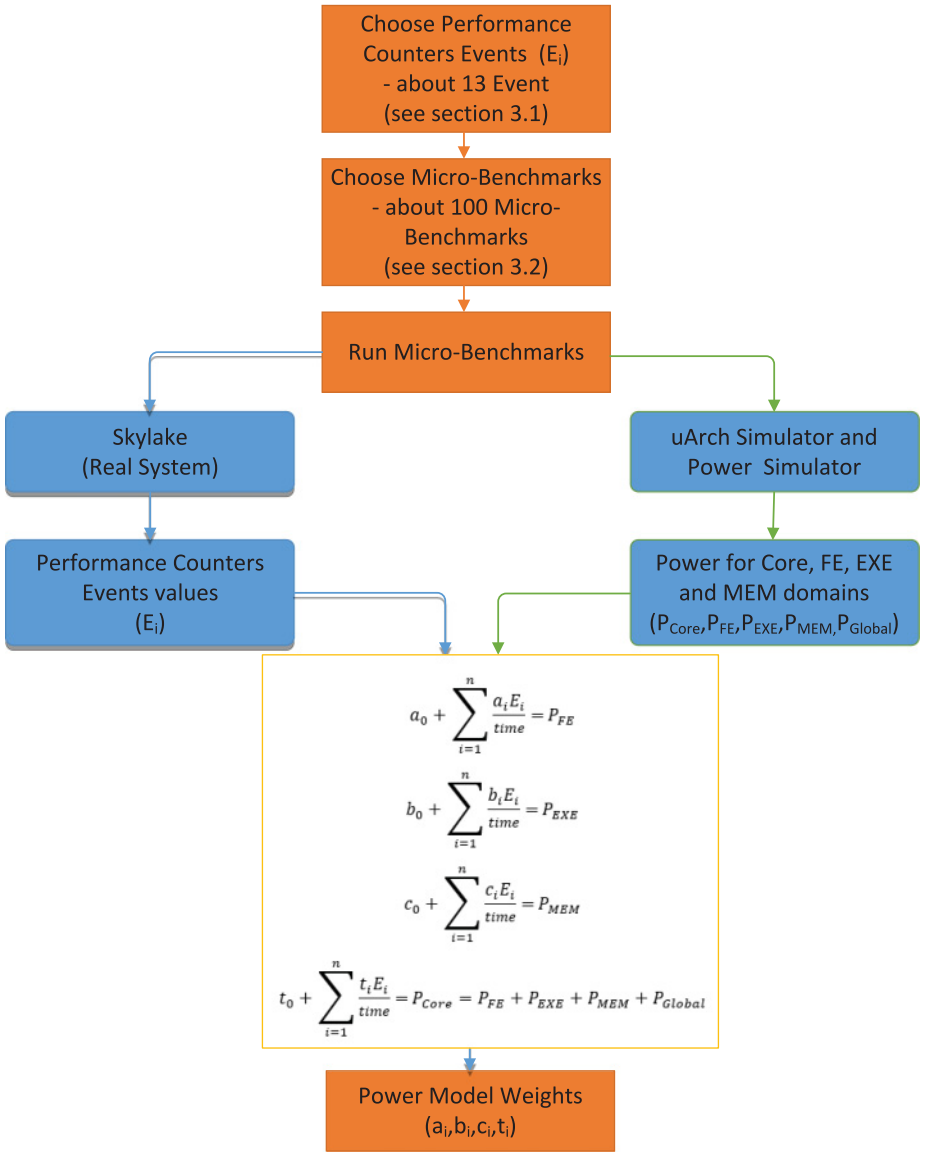


Fig. 2. Flow to determine the power-model weight based on simulator power-breakdown and performance-counters data.

system we were able to extract the values of the performance-counters that were generated for the specific micro-benchmark. Both the power and the counters' data were fed into Equation (2) in order to extract the values of the "weights" values after solving the linear regression. The flow for generating the power-model is described in Figure 2.

In this article, we will build a tool that estimates the power of the core and its sub-domains (Frontend, Execution, and Memory). It is important to note that there is a core power component that is not part of these sub-domains. This power consists of the power of global components ($P_{Global}$) such as clock-grid, power-gates leakage and others, this power is constant (at a given voltage and temperature), so the power of

the core ($P_{Core}$) is equal to the sum of the power of the sub-domains – Frontend ($P_{FE}$), Execution ($P_{EXE}$), Memory ($P_{MEM}$) – and the power of global components ($P_{Global}$).

### 3.1. Events Selection

While prior works [Isci and Martonosi 2003; Singh et al. 2009; Powell et al. 2009] adopted specific performance counters, we employed a process that started off an expert-driven selection of performance-monitoring events (counters) that was then empirically pruned using an iterative process until the model average error is closer to our target average error of 5%. We also carried out sanity checks to make sure that the winning counters were reasonable per Skylake microarchitecture details. Table shows the final list of performance counters that were chosen by our method.

*Initial Counters*. Set of initial counters (23 counters) were chosen based on an expert knowledge of the architecture and the available performance counters that the processor offers. These counters may be divided into three categories:

(i) Counters at *inbound/outbound of a domain*; for example, the counter UOPS_DISPATCHED_PORT.PORT_2 counts all load/store operations received at inbound of Memory domain while L2_RQSTS.MISS counts memory requests sent to off-core at same domain's outbound.
(ii) Coverage for *sub-units inside a domain*; for example, the counters LSD.UOPS and IDQ.{DSB,MITE,MS}_UOPS represent activity of the assorted fetch-units in the Frontend domain.
(iii) Counters *commonly used by prior work* such as cycle or Uops Retired.

Obviously, the counters' space is limited to the list published by Intel in the Skylake Core Performance Monitoring Unit (PMU), which entitles 157 raw counters in total [Intel 2015]. Many of these do not apply to our study and were consciously de-selected. For example, 80 out of the 157 do not apply to our scope – e.g. deal with Transactional Memory (TSX [Intel 2016a]), or count off-core related events like L3 miss, HW prefetches, large TLB pages, Hyper-threads interaction, or aliases to other existent events. Another group of 22 counters deal with issues that are less significant for the SPEC CPU2006 benchmarks – e.g., TLB flushes or misses in all levels or instruction fetch misses related. We carefully selected the 23 counters out of the remaining 55 counters. For example, 13 counters provide breakdown of L2 cache misses into per-request type; hence, we manually selected an aggregated: L2_RQSTS.MISS. We did kept granular events that are likely to affect the inside Core power consumption like in UOPS_DISPATCHED_PORT.PORT_$*$.

*Supervised Pruning*. Solving the linear regression was an iterative process. At each step, we dropped one counter – the least to impact power estimation accuracy. We verified that dropping each counter was reasonable when relating to its functionality. For example, UOPS_DISPATCHED_PORT.PORT_0 counts ALU operations executed, so it made sense to be included in the Execution domain and not in Memory domain. Each of UOPS_DISPATCHED_PORT.PORT_{2,3} count load/store operations in a symmetric fashion. The linear regression indicated only one counter is significant for the Memory domain (i.e., dependent variables). This can be verified as the two counters reported close values in all training micro-benchmarks.

*Sanity Checks*. We verified the winning counters are proper for each domain:
*In Frontend*. A counter was selected for each Fetch-unit as explained above. We can reason selection of UOPS_RETIRED and UOPS_ISSUED to quantify speculation impact since the Frontend domain is at the first stage of the pipeline making it most sensitive to speculation compared to other domains. UOPS_DISPATCHED_PORT.PORT_6

counts branches which also impact Frontend behavior, for example, lookup in Branch Prediction structures.

*In Memory*. UOPS_DISPATCHED_PORT.PORT_2 and L2_RQSTS.MISS at inbound and outbound of the domain were selected as expected. UOPS_DISPATCHED_PORT.PORT_6 was included since it well correlates with high-IPC (probably as a proxy for memory accesses hitting in the L1 data-cache).

*In Execution*. Ports 0 and 1 generally accommodate similar execution units such as integer ALU and FP arithmetic [Intel 2014]. UOPS_DISPATCHED_PORT.PORT_1 was selected in the process; PORT_0 was very close hence dropped. The FP_ARITH_$*$ counters provide scalar vs. vector (width) vs. precision distinction also made sense. Note also that the wider the vector size, or the higher the FP precision, the bigger the cost assigned per operation.

### 3.2. Training-Set Selection

A set of over hundred kernels was carefully selected for the linear regression. The designated collection of kernels has good coverage for both the architecture and the microarchitecture. It includes integer, floating point (single and double precision), vector (SSE, AVX, and AVX2 [Firasta et al. 2008]), and FMA instructions, crossed with various microarchitecture parameters such as cache miss rate. We also added kernels that exercise specific units of the microarchitecture such as the assorted fetch units.

The process of selecting the micro-benchmarks for the training set is similar to the events selection process that we described on previous subsection. The process started off an expert-driven selection of micro-benchmarks that was then empirically pruned using an iterative process. The micro-benchmarks were written in order to cover both the architecture and the microarchitecture feature and domains of the Skylake processor. The coverage of the features was monitored using the events that were selected previously. If some event was zero, then we wrote dedicated micro-benchmarks to cover it, for example the LSD.UOPS counters were zeros at the first set of the training micro-benchmarks, we wrote a dedicate micro-benchmark with a short loop that inters inside the LSD micro-operations cache and covered this event.

### 3.3. Reference and Estimated Power Calculation

We used an Intel proprietary simulator to obtain the reference power numbers for the core and its sub-domains. The simulator is built using micro-architecture events, these are obtained from a micro-architecture simulator (uArch-Sim) that runs before the power-simulator; the power-simulator uses the micro-architectural events and low-level design data (such as capacitance, or design block activity factor). The simulator uses power estimates from previous CPUs and circuit simulations on micro-benchmarks to estimate the power of architectural events (e.g., register access, cache access, ALU operation) and functional-unit-block (FUB level). The power per each event is multiplied by the activity of the block in order to estimate the power. Core sub-domain, such as FE, has many FUBs so to estimate the power of the Frontend (for example) the power-simulator uses hundreds of events. Summing all Frontend sub-domain FUBs power will give the Frontend total power. The accuracy of the Intel power proprietary simulator is 3–5% for performance workloads (i.e., workloads without CPU idleness)

For each counter, we assigned a weight in the equation to calculate the core domain's power. Equation (2) shows the linear function used, the $\text{PerfCounter}_i$ is the value of the performance-counter normalized by the sampling interval, the sampling interval is obtained from the ref_cycles fixed performance counter, dividing the ref_cycles by the reference-clock frequency (fixed clock and not part of the DVFS domain) gives

Table II. Performance Counter Used to Estimate the Core and Domains Power

| Domain | performance counter | weight |
|---|---|---|
| Core | Intercept | 541.7 |
| | FP_ARITH_RETIRED.128B_PACKED_DOUBLE | −6.008 |
| | FP_ARITH_RETIRED.256B_PACKED_DOUBLE | 234.207 |
| | FP_ARITH_RETIRED.256B_PACKED_SINGLE | 68.73 |
| | UOPS_DISPATCHED_PORT.PORT_1 | 286.368 |
| | UOPS_DISPATCHED_PORT.PORT_2 | 325.855 |
| | UOPS_DISPATCHED_PORT.PORT_6 | 150.967 |
| | IDQ.DSB_UOPS | 44.696 |
| | IDQ.MITE_UOPS | 164.858 |
| | IDQ.MS_UOPS | 218.486 |
| | LSD.UOPS | −14.001 |
| | UOPS_RETIRED.RETIRE_SLOTS | −26.606 |
| | L2_RQSTS.MISS | 1302.42 |
| | UOPS_ISSUED.ANY | 199.259 |
| Frontend | Intercept | 87.54 |
| | UOPS_DISPATCHED_PORT.PORT_2 | −84.597 |
| | UOPS_DISPATCHED_PORT.PORT_6 | 107.628 |
| | IDQ.DSB_UOPS | 18.1747 |
| | IDQ.MITE_UOPS | 136.256 |
| | IDQ.MS_UOPS | 213.617 |
| | LSD.UOPS | −77.274 |
| | UOPS_RETIRED.RETIRE_SLOTS | −82.495 |
| | UOPS_ISSUED.ANY | 157.881 |
| Execution | Intercept | 203.014 |
| | UOPS_DISPATCHED_PORT.PORT_1 | 158.128 |
| | FP_ARITH_RETIRED.128B_PACKED_DOUBLE | 63.0428 |
| | FP_ARITH_RETIRED.256B_PACKED_DOUBLE | 239.818 |
| | FP_ARITH_RETIRED.256B_PACKED_SINGLE | 84.2479 |
| | UOPS_ISSUED.ANY | 193.634 |
| Memory | Intercept | 127.209 |
| | UOPS_DISPATCHED_PORT.PORT_2 | 263.475 |
| | UOPS_DISPATCHED_PORT.PORT_6 | 78.8816 |
| | L2_RQSTS.MISS | 1426.51 |

the actual run time. Each normalized performance-counter value is multiplied by the $\text{weight}_i$ corresponding to the counter.

## 4. POWER ESTIMATION TOOL

Our tool estimates the power for the core and breaks it into three sub-domains of the core: Frontend (FE), Execution (EXE), and Memory (MEM). The counters used to calculate the power of each one of the components are different: these counters were determined according to the process described in the previous section and they showed the closest correlation to the components' power. The counters for each component are listed in Table II. Note that some counters have negative weight, this is due to overlapping (e.g., shared resources) in sub-domains that different counters target.

To collect the performance counter, we used Linux perf [Carvalho 2010]; this handles the configuration of the performance counter and can report its value. Besides, it features counter-event multiplexing simplified profiling when the number of performance events is higher than the physical counters supported at the processor.

We used a set of representative micro-benchmarks that stress and train each part of the core domains. We provide some details in Section 3.1.

### 4.1. Tool Auto-Calibration

The Intel simulator does not supply the absolute power number for the running micro-benchmark, it rather gives output that represents the dynamic-capacitance ($\alpha C_L$ in Equation (1) also denoted $C_{dyn}$) that each domain showed while running the micro-benchmark. To convert that value to real power, we need to know other parameters like voltage and frequency. The frequency is available to us but the voltage level is not visible to the user and it varies from one processor package to another. To extract the calibration values, we used the RAPL output which is already calibrated at the manufacturing process to report the correct power value. We did the calibration using a micro-benchmark that does not have L2 misses to make the RAPL reporting more accurate. The calibration is carried out by running the micro-benchmark on the simulator which reports $C_{dyn}$. We then ran the micro-benchmark on the real Skylake system where the micro-benchmark runs many times in a loop in order to have higher accuracy of the RAPL. The RAPL value is read and divided by run time to estimate what is the actual average power that was consumed by the micro-benchmark. We then divide that average power by the $C_{dyn}$ to get the *scaling-factor* that will be used to convert $C_{dyn}$ inside our tool into actual power.

The *scaling-factor* is machine specific as it depends on the voltage-frequency curve of the specific processor. Our tool carries out the calibration process automatically and estimates the *scaling-factor* at the tool initialization stage. This provides a significant advantage where our tool can auto-run on any product proliferation (aka SKU) based on the Skylake microarchitecture, not necessarily the specific system we experimented with, e.g., Desktop or Client models, Core i7, or Core m5, any frequency, etc.

### 4.2. Model Accuracy Validation

While we have high confidence in our breakdown model; it is not easy to find a reference model for the power of fine-grain components on real systems. To compare our tool with larger programs such as SPEC, we cannot use the power-simulator. Instead, this subsection recaps on various methods we used to verify the model accuracy.

Validating the model accuracy requires reference power model for the core and its sub-domains. We have used two reference models for this purpose, the first uses the Intel proprietary power simulator for the breakdown validation, while running on short programs (kernels), and the second reference model is the RAPL [David et al. 2010] energy reporting register for the core energy (PP0_ENERGY_STAUS MSR), while running on larger programs such as SPEC.

*4.2.1. Accuracy Vs. Power-Simulator.* We have validated our model and tool for power breakdown against the power-simulator, the process of the validation is described in Figure 3. A set of micro-benchmarks (listed in Table III) were used, each benchmark ran on the power simulator which outputs two sets of data:

(1) *Trace with Micro-Architectural Events*. The Intel proprietary power simulator uses the micro-architectural events and low-level design data (such as capacitance, design block activity factor, etc.) in order to estimate the power at sub-FUB levels.
(2) *Performance Counters*. The counter values are consumed by our power-model in order to estimate the power per each domain.

The results are shown in Figure 4, both the power-simulator (Sim) and the predicted power by our tool (Pred) are shown. In addition, the figure includes the average and maximum error across all domains for each micro-benchmark. We can see that the

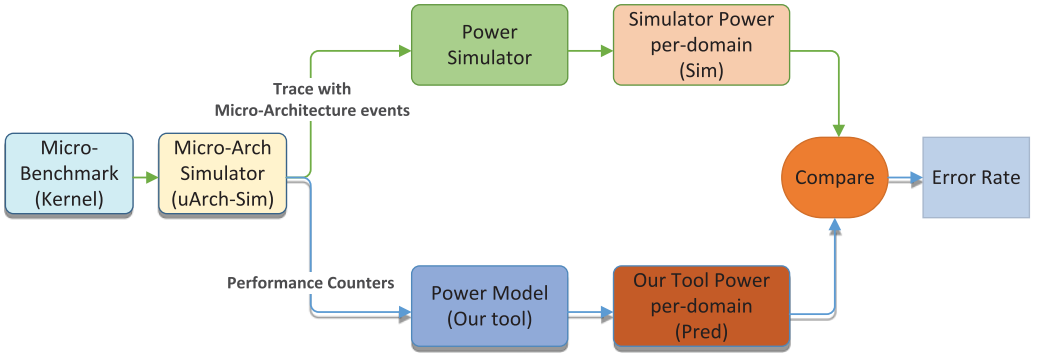


Fig. 3. Power-model validation process.

Table III. Micro-Benchmarks (Kernels) Used for Validating the Power Model

| Kernel Name | Kernel description |
|---|---|
| DAXPY | Double-precision a*X + Y kernel. Commonly used in matrix multiplication. |
| Mulss | Multiply Scalar Single-Precision Floating-Point instruction in a loop. |
| Spline_int_128bit | Spline interpolations (128bit) |
| Divpd | Packed Double-Precision Floating-Point Divide instruction in a loop stressing latency aspect. |
| DGEMM_MKL_256bit | Multiplying matrices using the dgemm routine at Intel Math Kernel Library (MKL), which calculates the product of double precision matrices |
| Sqrtpd | Compute Square Roots of Packed Double-Precision Floating-Point instruction stressing latency aspect. |
| UintToDouble_256 | Converts vector unsigned integer elements into double using vector instructions (256 bits) |
| Transpose_128bit | Matrix transpose using 128bits vectors |
| Add_Two_vec_256bit | Add two vector instructions of 256 bits |

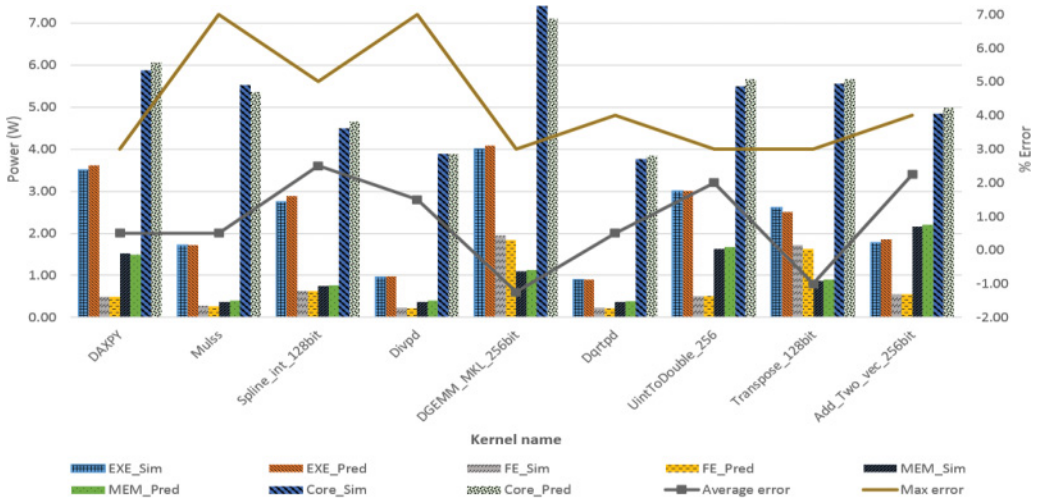


Fig. 4. Power per-domain outputted by power-simulator (_Sim) and our tool (_Pred) on various micro-benchmarks.

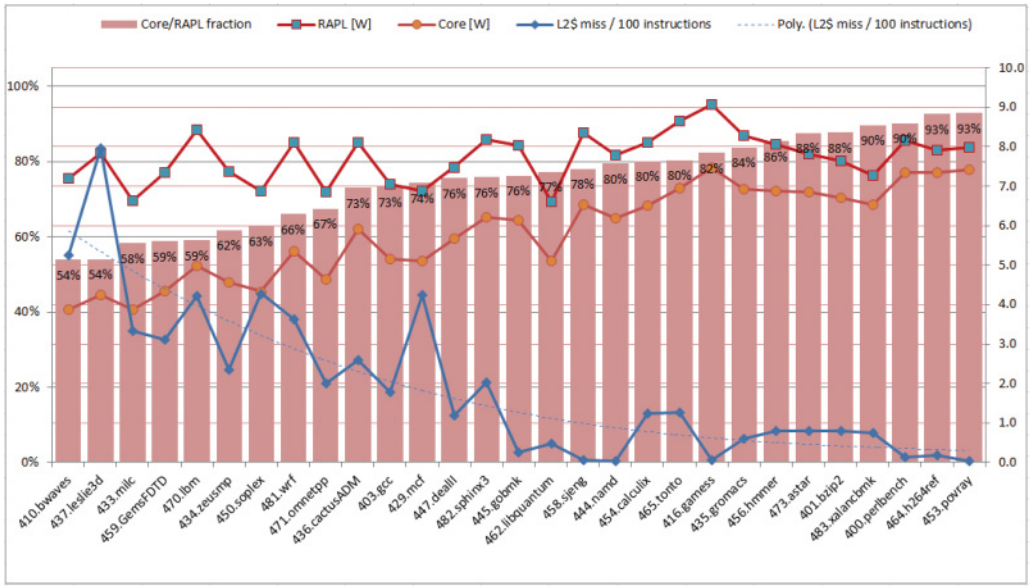


Fig. 5. RAPL power vs. core estimated power.

error rate is up to 7% compared to the Intel proprietary power simulator on the micro-benchmarks comparison. The error is due to multiple sources; this includes the linear regression inherited error. In addition, the data switching factor can lead to variations in the actual power consumption, hence, it's a source of error at our model.

*4.2.2. Accuracy vs. RAPL.* To compare our tool with larger programs such as SPEC, we cannot use the simulator, because running large programs on the simulator will require days of simulation time and very large storage space. For this purpose, we have validated the tool against RAPL [David et al. 2010] energy reporting register for the cores energy (PP0_ENERGY_STAUS MSR) on a real system. This MSR reports the energy consumed by the different cores and the LLC – it is the closest available to the granular levels our work models. Our tool does not attempt to model the LLC power (out of the Core domain). For validating the power consumption using RAPL, we looked at benchmarks that do not generate L2 cache misses and that work on single core. Results showed an error rate between 7% and 20% for benchmarks with near zero L2 cache misses. It is important to note that using RAPL to measure the power of a single core is not fully accurate; first, because the error rate of RAPL itself is up to 5% [David et al. 2010]; second, RAPL reading in our case estimates the power of all cores and LLC, if the cache is idle (no L2 cache misses) does not mean that its power is zero as the cache has static power and is not fully power-gated. Moreover, additional noise can arrive at RAPL reading from operating-system threads. If some OS thread wakes up, it can change the RAPL readings. Despite the RAPL inaccuracy, we wanted to show that our model correlates well with the RAPL reporting.

Figure 5 plots the Core-power from our model, RAPL PP0-power, alongside their ratio in bars for all SPEC CPU2006 INT and FP benchmarks sorted by the ratio. Additionally, we included the L2 cache miss rate and its trend (dashed line). We measure L2 cache misses of any request type. The bars use the left-hand Y-axis, while all others use the right-hand Y-axis. The graph suggests that the lower the L2 cache misses the closer is our Core estimated power to the RAPL "reference" power. The correlation between the two variables is 68% for all benchmarks and 75% for those with L2 cache miss rate below 0.5.

One source of inaccuracy we noticed for a subset of SPEC FP benchmarks: gromacs, games, namd, GemsFDTD, milc. These benchmarks have significant use of FP scalar (non-vector) operations – over 20% of retired uops are FP scalar compute operations – and so are likely to have their Execution domain and Core power underestimated.

*4.2.3. Performance Profile Cross-Check.* Last, we closely examined and cross-checked the power breakdown against performance breakdown (using the TMAM-validated method [Yasin 2014]) in addition to designated performance counters where applicable. This is extensively illustrated in the *5. Experiments and* Results section, for which a sizable part of this article was devoted.

### 4.3. Over-Time Estimation

Our tool can be configured using a command line option to report power-estimation over-time, the tool can report the average-power consumed every given time interval (e.g., 100ms, 1sec . . .). This option gives better understanding and visibility to the user about the behavior of the examined benchmark at different phases. A sample output is provided in Section 5.6.

### 4.4. The Scope of the Tool

The scope of this work is on single-threaded CPU-bounded workloads, i.e., performance workloads that do not include CPU idleness (C-states [Intel 2014]). Moreover, our tool assumes that the CPU performance state (P-states [Intel 2014]) and temperature do not change in the middle of the benchmarks runs, i.e., DVFS in the middle of the benchmark run is not supported. In addition, our tool uses auto-calibration methodology that enables porting it to different processors, frequencies and temperatures. Currently, a single frequency/temperature is supported during the benchmark run) without the need to re-calibrate manually if, for example, the voltage-frequency curve is changed.

The tool can be enhanced in the future to include multiple frequencies and temperatures by applying a calibration process at the initialization phase to multiple frequencies/temperatures and keep a mapping of scaling-factors per each frequency or temperature. This enhancement requires inquiring the processor frequency/temperature periodically and using the scaling-factor corresponding to the current frequency/temperature point.

The tool currently supports a single core and it can be extended to multi-core, as the performance-events reading today is per-core. Few changes are required during the calibration process and tool structure to support multi-core power-breakdown at sub-core domains.

## 5. EXPERIMENTS AND RESULTS

In this section, we present experimental results of the method on a Skylake system. First, we demonstrate how some compiler optimizations affect the breakdown on simple kernels. Then, we present characterization data for the SPEC CPU2006 benchmarks on our system with the setup described in Table III. We carry out a global analysis of the data and a drill down into a few case studies. Last, we share overall results on model accuracy.

### 5.1. System Setup Parameters

For the experiments in this study, we used a Skylake platform with the setup at Table IV. We also used pmu-tools/toplev [Kleen 2015] to obtain the full TMAM performance breakdown used in the case studies section.

*Temperature*. Preliminary experiments have shown that a rise in processor temperature can increase power by as much as 0.5 to 0.8 watts. To minimize this impact,

Table IV. Baseline System Setup Parameters

| | |
|---|---|
| **Processor** | Intel®Core™i7-6700K. 3GHz fixedfrequency, 8MB L3 cache. Hardware prefetchers disabled[1] |
| **Memory** | 8GB DDR4 @ 2133 MHz |
| **OS** | Ubuntu 14.04, Linux kernel 3.19, Hugepages disabled perf version 3.19.8-ckt5 |
| **Benchmark** | SPEC CPU 2006 v1.2 (base/rate mode) |
| **Compiler** | Intel Compiler 14 .AVX2 ISA with FMA. INT/FP target 32/64-bit modes, respectively |

[1]Due to erratum SKL095 [Intel 2016b] on OFFCORE_RESPONSE, the visibility to hardware prefetches is too limited to enable proper power modeling.

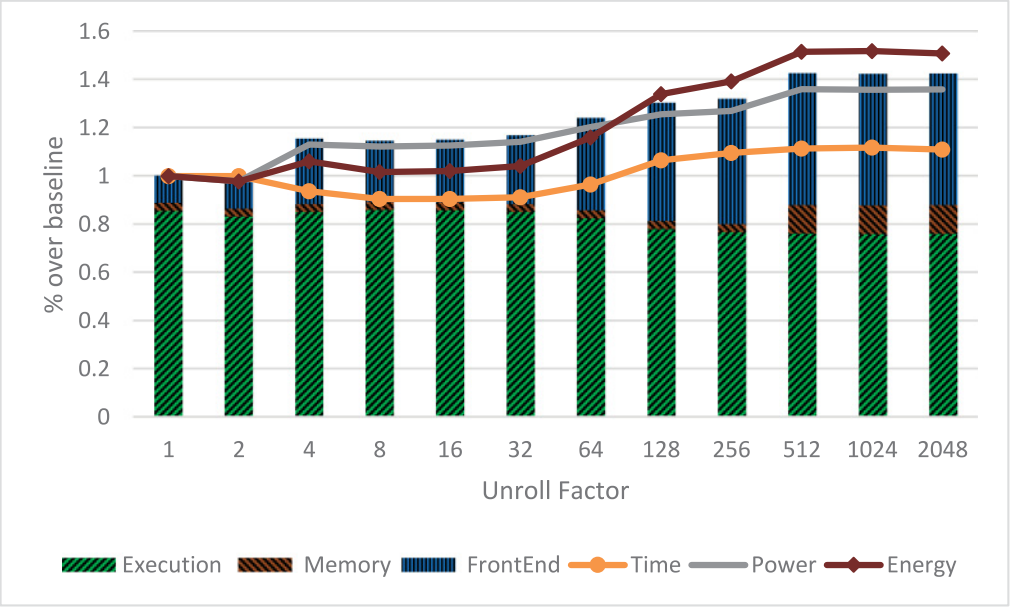


Fig. 6. Power breakdown and performance of unrolled Scalar ADD kernel.

benchmarks are run continuously for a five-minute warmup period to allow the temperature to stabilize before starting the experiment. We do not expect to have much variation on the temperature after the warm-up time as the workloads assumed to be CPU bounded and no idleness expected.

### 5.2. Sample Optimizations

We start with very simple kernels in order to reason our power modeling and see how common compiler optimizations affect power and performance.

Kernels (or micro-benchmarks) are tight loops of specific instructions. We choose different instruction types for implications on Execution domain power. Similarly, loop unrolling is used to check implications on the Frontend and Memory domains. Last, we examine the Fused Multiply Add (FMA) instruction which has been recently introduced by Intel's AVX2 and its efficiency in the Skylake implementation.

*5.2.1. Varying Code Size.* In this experiment, we built a micro-benchmark with simple loop of MOV and ADD register-to-register operation (no memory access). Then, we applied loop unrolling with factors of 1, 2, 4 . . . 2048 as shown in Figure 6.

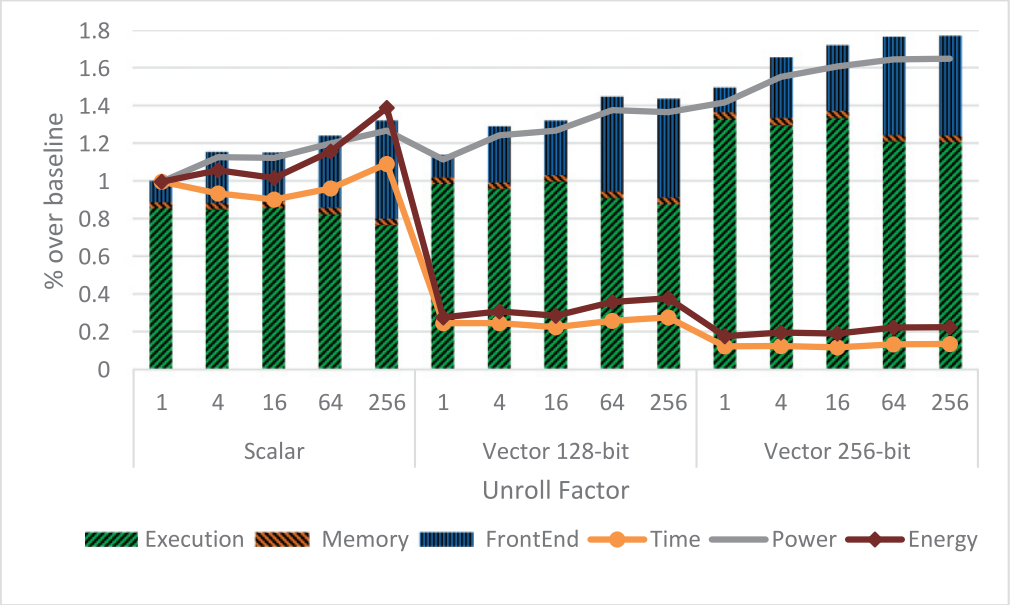


Fig. 7. Scalar, vector 128- and 256-bit versions of ADD kernel at different unrolling factors.

Figure 6 through Figure 8 report relative performance (Time), Power and Energy for the various unrolling factors in the X-axis. The bars are the per-domain power breakdown, and the total bar height represents the Core power relative to baseline (leftmost bar). The same annotation is used in figures later in this subsection.

The execution power, relative to baseline, is fairly constant as the unrolling factor increases. On the other hand, the Frontend power increases with a bigger unroll factor. The reason is that different fetch units (as depicted in Figure 1) deliver the instructions as the code size increases. For example, small loops can fit nicely in the Loop Stream Detector (LSD) which is power efficient as most of the other Frontend blocks are switched off [Intel 2014]. This explains the small Frontend power in the two most left bars. As the code size increases, it gets delivered by less efficient fetch units, power and performance wise. At unroll factor of 512 is where the code misses the L1 instruction cache (L1I) and fetches instruction lines from L2 cache. There is a noticeable power of the Memory domain which accounts for power consumption of the L2 cache accesses.

Overall, we can see a higher unroll factor (which resembles a bigger code size) drawing higher power while the performance also decreases. In this case, managing the unroll factor (or reducing code size) will give us optimization for power and performance together. That is, this example demonstrates that optimization for performance can be also optimization for power.

*5.2.2. Vectorization Impact.* In this example, we ran the same simple ADD-MOV micro-benchmark on scalar, SSE (128 bits) and AVX (256 bits) Instruction Set Architectures (ISA) [Intel 2014].

In Figure 7, we can see that the performance of SSE is better than the Scalar case while the AVX has the best performance. SSE improves performance by almost 4x since the kernel uses single-precision floating-point accuracy (32-bits). The power of AVX is 42–65% higher than baseline (Scalar) as expected since the vector operations consume more power (multiple operations vs. single operation in scalar). This is explained by the bigger Execution bars as we move from Scalar to Vector 128-bit to Vector 256-bit

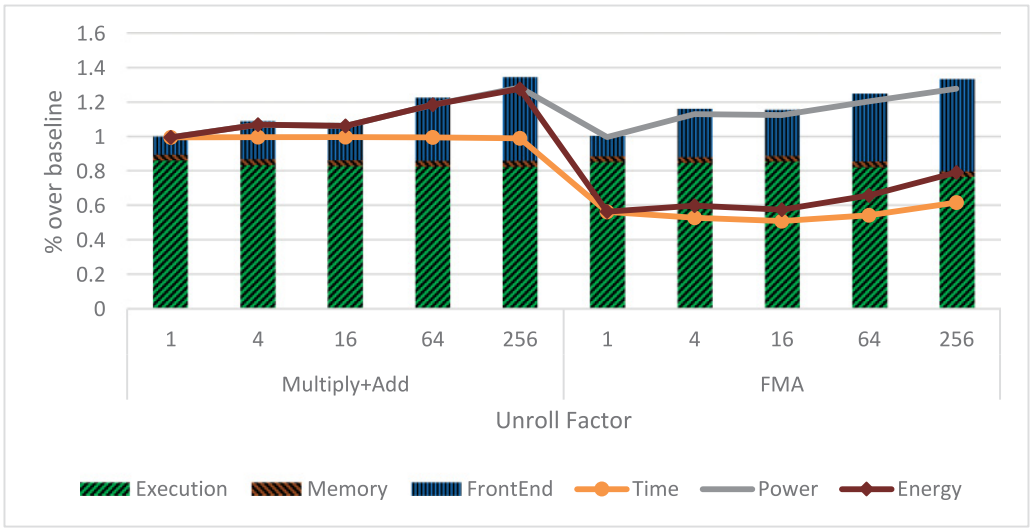


Fig. 8. Scalar Multiply-Add and FMA kernels at different unroll factors.

categories. The code size (unrolling factor) has a similar impact as we observed *in* Figure 6 which suggests the Frontend power works on more instructions set.

Note that overall Energy with AVX is better than the baseline. This demonstrates that while vector instructions can consume higher power, they are more energy efficient than non-vector scalar instructions. That is, vectorization is an energy-efficient optimization on the Skylake microarchitecture.

*5.2.3. The FMA New Instruction.* In this experiment, we took two micro-benchmarks with a simple loop that does Multiply-Add between registers (1 mov, 2 operations, no memory accesses). The first micro-benchmark does the Multiply-Add using two distinct instructions, while the second one uses the recently introduced FMA instruction [Intel 2016b]. Both kernels do same amount of work.

In Figure 8, we can see that the performance of the FMA is better by a factor of 2x than Multiply-Add using distinct instructions. The overall power has increased by ∼10% for the middle unroll factors. Hence, FMA has better net energy than a none-fused version.

When we look at a power breakdown, the Execution power is slightly better in the FMA case (or another way of looking at it is that the power of the distinct Multiply/Add is higher), possibly since fewer net operations are sent to the execution units. The code size (unrolling factor) has similar impact like we observed in Figure 6.

### 5.3. Floating Point Apps

We ran our model on SPEC CPU2006 FP benchmarks built with an Intel Compiler. The next two figures are twofold: the upper part represents a power profile with absolute power (solid lines in Watts) for the package and core as well as the breakdown inside the core (bars are fractions). The lower part is the performance profile with the absolute performance – Instructions Per Cycle (IPC) – and its breakdown using TMAM [Yasin 2014] with the representation is described in Section 2.5. The benchmarks are sorted in increasing Retiring left to right. Retiring is the fraction of pipeline slots utilized by operations that eventually retire (i.e., from the program's true path). Generally, the higher the Retiring bucket, the higher the IPC.

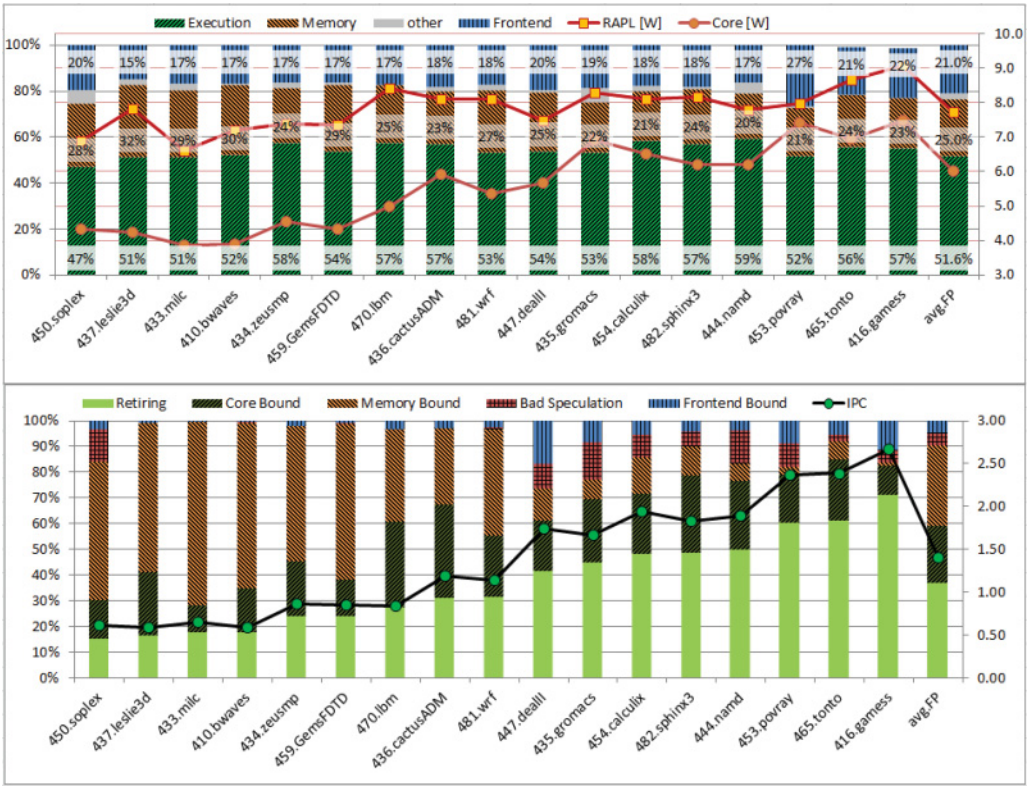


Fig. 9. Power (upper) and performance (lower) profiles of SPEC CPU2006 FP apps.

Notice the RAPL [in Watts] is retrieved based on RAPL PP0 counter (see Section 2.5) provided by the processor hardware (that is not relying on our method), while Core is modeled by our tool.

In Figure 9, the power lines in the upper graph correlate with the performance whenever the Core power is the greater part of the package power (RAPL). That is, *the general trend is that Core power correlates with IPC*. This makes sense as one would expect the higher the throughput of operations executed by a processor (IPC), the bigger the power consumed to do that. One exception to this trend is evident with the move from 447.dealII to 435.gromacs where the power increases while performance decreases. An inverse of this phenomenon occurs for 433.milc though to a lesser extent. We study both cases in Sections 5.5.1 and 5.5.2.

On average, the Core is the biggest power consumer within the package for these FP workloads where it consumes 71% of the power.

One interesting insight is that the Frontend consumes 21% of core power, while it is an insignificant performance bottleneck – Frontend Bound is 5% on average. This can be explained as the Frontend portion of the pipeline must always fetch instructions to feed the Backend, regardless whether the fetch operation itself is critical performance-wise or not. Since compiler's code generation heavily impacts the Frontend footprint, hopefully this means there is a room for energy efficiency. Profile-guided optimizations may be promising in this direction if they can reduce the power consumption of the Frontend without degrading performance. The LSD is one example hardware optimization as demonstrated in Section 5.5.3.

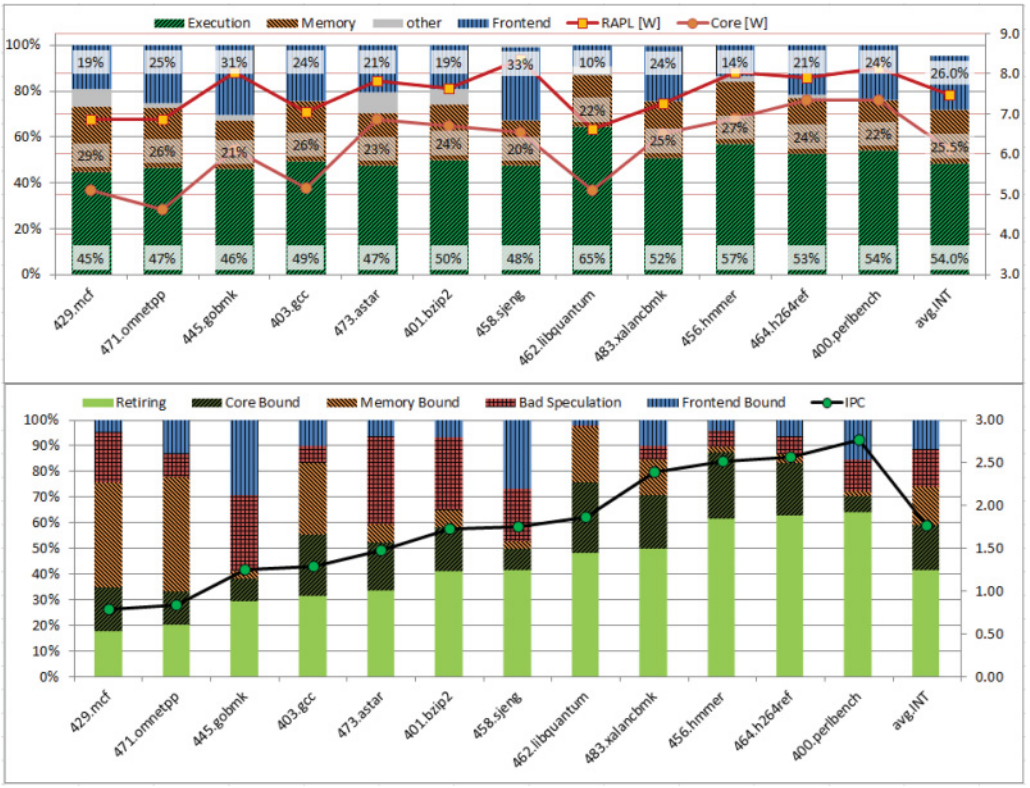


Fig. 10. Power (upper) and performance (lower) profiles of SPEC CPU2006 INT apps.

For benchmarks on the left side (whose IPC is below one), our model suggests Core power is not the significant part of package power. This corroborates with the performance profile where these benchmarks get low IPC since they are significantly Memory Bound. Examining toplev [Kleen 2015] results for next levels in the TMAM hierarchy, suggests the performance bottleneck is in areas outside the core – LLC and External Memory (we omitted these results due to space limits). Notice the lower core power fraction suggests the design is somewhat efficient where the cores do not burn much power while waiting for the off-core memory subsystem. In particular, we see the Memory power fraction for them is in the 24%–32% range of core power, which is not too far from the average (25%).

### 5.4. Integer Apps

The INT benchmarks show a similar *general trend where the Core power correlates with IPC* in Figure 10. See previous subsection for graph annotation. A key exception case is 462.libquantum which has a noticeable drop in power compared to its predecessor benchmark and yet has better performance – we study this case in Section 5.5.3.

On average, the core consumes 82% of the package power for the INT benchmarks (recall FP was 71%). This can be explained by examining the TMAM performance profile. INT benchmarks are less bounded on memory performance-wise (average Memory Bound is 15% for INT vs. 31% for FP). This means more of the power is likely to be consumed by components inside the core, akin to what was discussed for Memory Bound

FP benchmarks in previous section. Moreover, INT benchmarks have a higher fraction of Bad Speculation (15% vs 5%), which in turn means the core requires more power while executing operations out of an incorrect program path or when recovering back to the true path afterwards.

Looking at the power breakdown, INT benchmarks show higher Execution domain power fraction compared to their FP counterparts. This is most likely due to the higher average IPC (1.76 vs. 1.4). Also the Frontend power fraction is higher, since INT benchmarks are known to be more sensitive performance-wise on instruction fetch and branch predictions [Yasin 2014].

### 5.5. Case Studies

In this section, we examine some interesting cases of the SPEC CPU2006 results.

*5.5.1. 435.gromacs.* 435.gromacs consumes core power of 6.9W as reported by the data of Table VI in the Appendix. Figure 9 suggests this is a significant increase over its predecessor (447.dealII with 5.7W core power). On the other hand, the IPC has slightly decreased when comparing the same two benchmarks.

We use the Floating Point OPerations per Cycle (FLOP/C) metric to explain this. One can claim it is a better metric to represent FP performance than plain IPC, since it properly accounts for vectorization throughput. For example, a 256-bit vector ADD instruction can serve up to eight single-precision FP elements. It is counted as one in IPC vs. eight in FLOP/C.

Table VI reports the FLOP/C and IPC metrics for all FP benchmarks. Note FLOP/C is at 2.24 for 435.gromacs. This means while 435.gromacs executes instructions at a slightly lower rate compared to 447.dealII, it delivers FLOP/C at a much higher rate (by a factor over nine). Our Execution domain power model correctly reflects this case: it has higher absolute power of 3.4W in 435.gromacs vs 2.8W in 447.dealII (20% higher). Microarchitecture wise, the execution cluster is where the power consumption manifests when executing vectors vs. non-vector operations.

*5.5.2. 433.milc.* An inverse of the 435.gromacs phenomenon occurs for 433.milc: it has higher performance and lower power compared to its predecessor (437.leslie3d) in Figure 9. We use FLOP/C once more to explain this with data from Table VI in the Appendix.

While 433.milc has higher IPC, it actually has a lower FLOP/C metric (by a factor of almost 2) compared to 437.leslie3d. Examining the performance counters of the Skylake processor, we observe that only 8% of the arithmetic instructions of 433.milc has actually used vector ISA, compared to 95% for 437.leslie3d. The power model indicates less Execution and Core absolute power for 433.milc over 437.leslie3d (see Table VI). Again, our model correctly reflects this case, despite the fact that the Core power is not the biggest consumer of package power for these two benchmarks.

*5.5.3. 462.libquantum.* Recall that the general trend was that Core power correlates with IPC in Figure 10. 462.libquantum is a positive outlier where its IPC is slightly better than its predecessor benchmark (458.sjeng) while it has quite good Core power reduction (6.5W for 458.sjeng down to 5.1W for 462.libquantum per Table VI).

The power breakdown clearly shows the Frontend power was reduced from 33% (2.0W) down to 10% (0.4W). This has occurred since almost all of the instructions of 462.libquantum were actually delivered by the LSD unit which is the most efficient fetch unit in Frontend as indicated in Section 5.2.1. In comparison, only 1% of 458.sjeng instructions were delivered by the LSD.

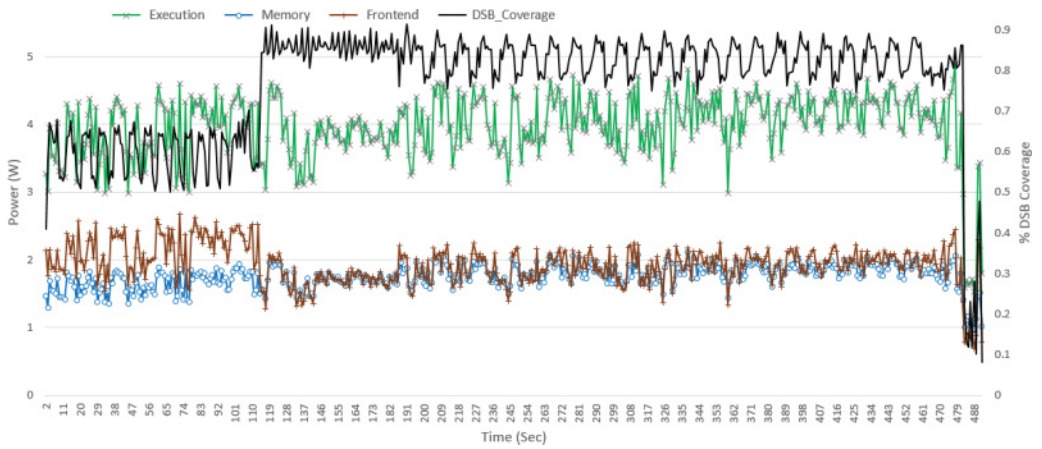


Fig. 11. Trace over time for the 416.gamess with 1000ms interval.

### 5.6. Trace Over Time

Our tool can be configured using command line option to report power-estimation over-time, the tool can report the average-power consumed every given time interval (e.g., 100ms, 1sec . . .). This option gives better understanding and visibility to the user about the behavior of the examined benchmark in different phases.

Figure 11 shows trace over time for the 416.gamess benchmark as outputted by our tool. The graph also includes the sampling over time of the Decoded Stream Buffer (DSB) coverage performance metric. The DSB serves as decoded uop-cache, that allows the Frontend to be run at lower power in case the uops stream out of the DSB. The figure shows that the Frontend power is relatively high at the interval of 0–110 seconds, while the power reduces at the middle interval that lasts till second 470. We can observe that the power of the Frontend is high when the DSB coverage is low and vice-versa.

## 6. RELATED WORK

Frank Bellosa's work [Bellosa 2000] shows the linear correlation of hardware events and energy. Then, they use hardware activity to establish a thread-specific energy consumption model. Since then, a lot of work has been done for power estimation based on hardware performance counters. Performance monitor counters (PMCs) based power estimation models in earlier work can be divided into two categories [Bertran et al. 2013a]. The top-down method uses a reduced set of events (usually around 5), aiming to build a fast and simple model with small overheads, while the bottom-up approach breaks down the power components based on micro-architecture (it can use hundreds of micro-architectural events). The latter produces a more accurate power model by gathering more information to reflect the power characteristics of applications, at the cost of increasing the complexity of modeling. Our model balances between accuracy and cost, and can be viewed as a moderation on both sides (we used about 13 events).

Bertran et al. [2010] present a decomposable power model for Intel Core 2 Duo processor. They use a total of thirteen counters for estimating the power consumption of different micro-architectural components. They use micro-benchmarks and incremental linear regression technique to come up with individual power consumption figures for components such as Frontend, Integer execution units, FP units, BPU, and SIMD units. A later work by Bertran et al. [2013b] reduces the problem of generating decomposable power models for different architectures by proposing a systematic modeling methodology. The method used by Bertran et al. [2010] for power measurement using

SMAPI [ThinkPad] is platform specific: it uses battery charging information to estimate power. This method cannot be applied for desktop and is less accurate relative to RAPL which measures the power for finger domains and not for the whole platform. In addition, the method of Bertran et al. [2010] has a restriction that it cannot map micro-architectural components to power components if the activity of the micro-architectural components cannot be isolated using dedicated micro-benchmarks; in our method, we use the power simulator data that have accurate power estimation per component. Moreover Bertran et al. [2010] does not have an auto-calibration capability; this means that when moving to another processor (same architecture) or applying a different frequency or temperature the user needs to regenerate the model for the new setup.

Powell et al. [2009] proposed a model that uses 9 PMCs to estimate the power of various core sub-domains of Intel-like core architecture; they used a power-simulator and an architecture-simulator to produce the model that was built using linear regression on the data obtained from the power and architecture simulators. The model reports pseudo-watts and not actual power. However, comparing with our tool, we have implemented the tool on real systems and our tool reports read power (in Watts). In addition, our tool does auto-calibration that enables porting it to another processor or running at different frequencies. Moreover, the model uses 9 PMCs to estimate the power of more than hundred core sub-units; this is less accurate than our model that uses a dedicated counter for each sub-unit that best represents it. The pseudo-power reported by the model from Powell et al. [2009] is compared to a simulator which is itself 5% to 10% accurate. Our tool runs on a modern real processor and was compared against the RAPL reporting.

Singh et al. [2009] achieves a run-time per-core power estimation of multithread and multi-program workloads using the top-down method [Bertran et al. 2013a]. The model estimates the power for each core at thread level, not at sub-core domains. They categorize the processor's hardware event into four classes (because their environment platform has only four performance counters). Then the topmost one is chosen in each class, which is the most correlated to power. With the runtime data from executing micro-benchmark, they built a piece-wise linear mode and achieved median errors of 3.9%, 5.8%, and 7.2% for the SPEC-OMP, NAS, and SPEC2006 benchmark suites, respectively. However, compared with ours, their model is estimating at lower-granularity (thread level) while we model the per-core domains. Furthermore, their model uses relatively few counters (four counters) while we use dedicated counters for each sub-domain of the core; we used 13,8,5,3 counters to model the power of the whole core, Front-end cluster, Execution cluster and Memory cluster respectively.

Isci and Martonosi [2003] decompose CPU into 22 power breakdowns based on a function unit, which is a typical bottom-up approach [Bertran et al. 2013a]. They estimated the maximum power for each of the units according to physical area based on Pentium 4 die photo. Following that, they present a per-unit power estimation devised from performance counters. They train the sub-model with a set of specialized micro-benchmarks to stress the correlated power units one by one. Many hardware events can reflect more power characteristics, making a contribution to the predicted accuracy. We treat things as a whole and combine different units on a higher level view to avoid a cumbersome per-unit model. In addition, our tool uses auto-calibration methodology that enables porting it to different processors without the need to re-calibrate manually if, for example, the voltage-frequency curve is changed.

Shao and Brooks [2013] propose to characterize workloads independent of the ISA (Instruction Set Architecture). They profile memory, branch and opcode behavior of an application using the intermediate representation of an application instead of the binary. Following up on this work, they propose using the ISA-independent profiles for

Table V. Comparison of Recent Power Estimation Methods

| Work | Scope | | Accuracy | | Granularity | Reporting |
|---|---|---|---|---|---|---|
| **Bertran et al. [2010]** | System | Intel Core 2 Duo | ~6% | Calibrated using SMAPI[Think Pad] | Medium-High: Dedicated per group of micro-architectural components | Power in watts |
| **CAMP Powell et al. [2009]** | Simulation | (Intel Core 2009) | 10-15% | Calibrated against ALPS propriety model | High: Dedicated per-unit | Pseudo power [Pseudo-watts] |
| **Isci et al. [2003]** | System | (Pentium 4, 2003) | ~ 13% | | High: Dedicated per-unit | Power in watts |
| **Spiliopoulos et al. [2012]** | System | (Nehalem 2008) | 5-15.1% (average 3.9%) | | Very Low (No breakdown): whole CPU power | Power in watts |
| **RAPL [David et al. 2010]** | System | (Sandy Bridge 2011) | 5% | | Low (Coarse): Cores+LLC, Graphics, platform, DDR | |
| **Ours** | System | (Intel Core, Skylake 2015) | 5-10% | Calibration with RAPL | Medium: Core sub-domain (group of units) | Power in watts |

speeding up the design of accelerators [Shao et al. 2014]. They achieve good speedups with minimal loss in accuracy both for performance and power estimations. In addition, Van den Steen et al. [2015] demonstrated an analytical performance and power model, based on micro-architecture independent application profiles to enable the evaluation of large design spaces using a single application-specific but architecture-independent profile. Performance and power are estimated with an average error of 13% and 7%, respectively.

Table V provides a high-level comparison of recent power estimation methods that are performance-counter-based and outlines the key differences among these methods when compared to our methods described in this article. The reference of each of the prior methods has the details.

## 7. SUMMARY AND FUTURE WORK

In this work, we proposed a new methodology that combines power, performance, and thermal measurements, together with selected performance counters to provide accurate low-level power estimations. The method was demonstrated on a modern out-of-order core including a breakdown to its key sub-domains: Frontend, Execution, and Memory. A tool that implements the methodology is built and calibrated for the Skylake processor – newly released Intel. We demonstrated how the power estimation and per-domain breakdown reflect the expected behavior for a few hand-crafted micro-benchmarks as well as when select software optimizations are used: vectorization, loop unrolling and the new FMA instruction.

This work provides a detailed power and performance characterization breakdown for the SPEC CPU2006 benchmarks as measured on the Skylake system. While our detailed analysis substantiates the general trend that the core power correlates with performance (IPC), the details are important. For example, FLOP/cycle is a better FP efficiency metric than plain IPC – both metrics are included in the report. We have also found that the Frontend consumes a sizable fraction of the core power even when it is not a performance limiter which hopefully suggests room for software and hardware optimizations.

For future work, we plan to extend the tool to model other processor cores which can help to evaluate and understand efficiency of optimizations at a finer-grain level. We also plan to enhance the tool to provide hints to software developers and guide the user to catch power bottlenecks in the programs similar to performance hints given by the Top-down Microarchitecture Analysis Method.

## APPENDIX

Table VI provides a power and performance characterization report for the SPEC CPU2006 benchmarks. The remaining power numbers are modeled by our tool. Note RAPL.PP0 accounts for off-core power as well (e.g., LLC), the RAPL.PPL0 will be close to the core-power once the L2-cache-miss is close to 0 and only one core is working. See Table IV for system setup details. In addition, note that the sum core power breaks down to Frontend, Execution, Memory and Global, the $P_{Global}$ is constant does not appear on the table, its value is about 0.6 watts.

Table VI. SPEC CPU2006 Power and Performance Characterization Data on Skylake

| | Power [W] | | | | | Performance | |
|---|---|---|---|---|---|---|---|
| **Benchmark** | ***RAPL/PP0*** | **Core** | **Front-end** | **Execution** | **Memory** | **IPC** | **FLOP /cycle** |
| 400.perlbench | *8.2* | 7.4 | 1.6 | 3.7 | 1.5 | 2.76 | |
| 401.bzip2 | *7.6* | 6.7 | 1.2 | 3.1 | 1.5 | 1.73 | |
| 403.gcc | *7.0* | 5.2 | 1.1 | 2.3 | 1.2 | 1.29 | |
| 410.bwaves | *7.2* | 3.9 | 0.6 | 1.8 | 1.0 | 0.58 | 0.47 |
| 416.gamess | *9.1* | 7.5 | 1.6 | 3.9 | 1.6 | 2.66 | 0.79 |
| 429.mcf | *6.9* | 5.1 | 0.9 | 2.0 | 1.3 | 0.78 | |
| 433.milc | *6.6* | 3.9 | 0.6 | 1.7 | 1.0 | 0.65 | 0.44 |
| 434.zeusmp | *7.4* | 4.6 | 0.7 | 2.3 | 1.0 | 0.86 | 1.42 |
| 435.gromacs | *8.3* | 6.9 | 1.2 | 3.4 | 1.4 | 1.66 | 2.24 |
| 436.cactusADM | *8.1* | 5.9 | 1.0 | 3.1 | 1.3 | 1.18 | 2.75 |
| 437.leslie3d | *7.8* | 4.2 | 0.6 | 1.9 | 1.2 | 0.58 | 0.86 |
| 444.namd | *7.8* | 6.2 | 0.9 | 3.4 | 1.1 | 1.88 | 1.03 |
| 445.gobmk | *8.0* | 6.1 | 1.7 | 2.6 | 1.2 | 1.25 | |
| 447.dealII | *7.5* | 5.7 | 1.0 | 2.8 | 1.3 | 1.74 | 0.25 |
| 450.soplex | *6.9* | 4.3 | 0.7 | 1.8 | 1.1 | 0.62 | 0.07 |
| 453.povray | *8.0* | 7.4 | 1.8 | 3.6 | 1.5 | 2.37 | 0.52 |
| 454.calculix | *8.1* | 6.5 | 1.1 | 3.5 | 1.3 | 1.94 | 1.63 |
| 456.hmmer | *8.1* | 6.9 | 0.9 | 3.6 | 1.7 | 2.51 | |
| 458.sjeng | *8.4* | 6.5 | 2.0 | 2.9 | 1.2 | 1.75 | |
| 459.GemsFDTD | *7.4* | 4.3 | 0.6 | 2.1 | 1.1 | 0.84 | 0.56 |
| 462.libquantum | *6.6* | 5.1 | 0.4 | 3.0 | 1.0 | 1.86 | |
| 464.h264ref | *7.9* | 7.3 | 1.5 | 3.6 | 1.7 | 2.56 | |
| 465.tonto | *8.7* | 7.0 | 1.4 | 3.6 | 1.5 | 2.39 | 0.92 |
| 470.lbm | *8.4* | 5.0 | 0.8 | 2.6 | 1.1 | 0.83 | 1.57 |
| 471.omnetpp | *6.9* | 4.6 | 1.0 | 1.9 | 1.1 | 0.84 | |
| 473.astar | *7.8* | 6.9 | 1.3 | 3.0 | 1.5 | 1.48 | |
| 481.wrf | *8.1* | 5.4 | 0.9 | 2.6 | 1.3 | 1.14 | 2.09 |
| 482.sphinx3 | *8.2* | 6.2 | 1.0 | 3.3 | 1.3 | 1.83 | 0.94 |
| 483.xalancbmk | *7.3* | 6.5 | 1.5 | 3.1 | 1.5 | 2.40 | |

## ACKNOWLEDGMENTS

The authors would like to acknowledge Hamid Reza Ghasemi, members of the Multifacet research group, and our anonymous reviewers for their helpful comments. We would like to thank the anonymous reviewers for their constructive feedback. We thanks Intel for providing access to the Skylake system and simulators data as well as for Tal Katz for lab support. We reserve special thanks to Andi Kleen for his invaluable help with performance monitoring in Linux.

## REFERENCES

F. Bellosa. 2000. The benefits of event: Driven energy accounting in power-sensitive systems. In *Proceedings of the 9th Workshop on ACM SIGOPS European Workshop: Beyond the PC: New Challenges for the Operating System*. ACM, 37–42.

R. Bertran, M. Gonzalez, X. Martorell, N. Navarro, and E. Ayguade. 2010. Decomposable and responsive power models for multicore processors using performance counters. In *Proceedings of the 24th ACM International Conference on Supercomputing*. ACM, 147–158.

R. Bertran, M. Gonzàlez, X. Martorell, N. Navarro, and E. Ayguadé. 2013a. Counter-based power modeling methods: Top-down vs. bottom-up. *The Computer Journal* 56, 2, 198–213.

R. Bertran, M. Gonzalez, X. Martorell, N. Navarro, and E. Ayguade. 2013b. A systematic methodology to generate decomposable and responsive power models for CMPs. *IEEE Transactions on Computers* 62, 7, 1289–1302.

S. Bhunia, S. Mukhopadhyay. (eds.). 2010. *Low-power variation-tolerant design in nanometer silicon*. Springer-Verlag.

A. Carvalho. 2010. The new linux 'perf' tools. *Presented at the Linux Kongress*, 2010. https://scholar.google.co.il/scholar?q=The+new+linux+perf+Carvalho.&btnG=&hl=en&as_sdt=0%2C5.

H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, and C. Le. 2010. RAPL: Memory power estimation and capping. In *2010 ACM/IEEE International Symposium on Low-Power Electronics and Design (ISLPED)*. IEEE, 189–194.

N. Firasta, M. Buxton, P. Jinbo, K. Nasri, and S. Kuo. 2008. Intel AVX: New frontiers in performance improvements and energy efficiency. *Intel White Paper*.

J. Haj-Yihia, Y. B. Asher, E. Rotem, A. Yasin, and R. Ginosar. 2015. Compiler-directed power management for superscalars. *ACM Transactions on Architecture and Code Optimization (TACO)* 11, 4, 48.

Jawad Haj-Yihia, Ahmad Yasin, Yosi ben Asher, and Avi Mendelson. 2016. Core Power breakdown tool. https://drive.google.com/open?id=0B3IgzCqRS5Q_ZGN0QVFqaWxxY28.

Intel Corporation. 2014. Intel® 64 and IA-32 Architectures Optimization Reference Manual, Appendix B.1 Intel. (as of August 2014).

Intel Corporation. 2015. "Intel open source", online: http://download.01.org/perfmon/ [accesses October 8, 2015].

Intel® 64 and IA-32 Architectures Software Developer's Manual. 2016a. Volume 3A: System Programming Guide, Part 1, [accesses January 2016a].

Intel Corporation. 2016b. "6th Generation Intel® Processor Family – Specification update", online: http://www.intel.com/content/www/us/en/processors/core/desktop-6th-gen-core-family-spec-update.html [accesses August 2016].

C. Isci and M. Martonosi. 2003. Runtime power monitoring in high-end processors: Methodology and empirical data. In *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture,* 93. *IEEE Computer Society.*

C. Isci, A. Buyuktosunoglu, C. Y. Cher, P. Bose, and M. Martonosi. 2006. An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 347–358.

A. Kleen. 2015. Toplev manual (pmu-tools), online: https://github.com/andikleen/pmu-tools/wiki/toplev-manual [accesses October 8, 2015].

M. D. Powell, A. Biswas, J. S. Emer, S. S. Mukherjee, B. R. Sheikh, and S. Yardi. 2009. CAMP: A technique to estimate per-structure power at run-time using a few simple parameters. In *2009 IEEE 15th International Symposium on High Performance Computer Architecture*. IEEE, 289–300.

R. Efraim, R. Ginosar, C. Weiser, and A. Mendelson. 2014. Energy aware race to halt: A down to EARtH approach for platform energy management. *IEEE Computer Architecture Letters* 13, 1, 25–28.

E. Rotem, A. Naveh, A. Ananthakrishnan, E. Weissmann, and D. Rajwan. 2012. Power-management architecture of the intel microarchitecture code-named sandy bridge. *IEEE Micro* 2, 32, 20–27.

Y. S. Shao and D. Brooks. 2013. ISA-independent workload characterization and its implications for specialized architectures. In *Proceedings of the International Symposium on Performance Analysis of Systems and Software (ISPASS 2013)*, 245–255.

Y. S. Shao, B. Reagen, G. Y. Wei, and D. Brooks. 2014. Aladdin: A preRTL, power-performance accelerator simulator enabling large design space exploration of customized architectures. In *Proceedings of the 41st Annual International Symposium on Computer Architecture (ISCA)*, 97–108.

K. Singh, M. Bhadauria, and S. A. McKee. 2009. Real time power estimation and thread scheduling via performance counters. ACM SIGARCH *Comput. Architect. News* 37, 2, 46–55.

THE GREEN500 SITES. 2013. http://www.green500.org (accessed December 12, 2013)

ThinkPad SMAPI kernel module version 0.40. http://tpctl.sourceforge.net/.

TOP 500 SUPERCOMPUTER SITES. 2013. http://www.top500.org/list/2013/06 (accessed December 12, 2013)

Vasileios Spiliopoulos, Andreas Sembrant, and Stefanos Kaxiras. 2012. Power-sleuth: A tool for investigating your program's power behavior. In *Proceedings of the 2012 IEEE 20$^{th}$ International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE.

S. Van den Steen, S. De Pestel, M. Mechri, S. Eyerman, T. Carlson, L. Eeckhout, E. Hagersten, and D. Black-Schaffer. 2015. Micro-architecture independent analytical processor performance and power modeling. In *Proceedings of the 2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, Mar. 2015

A. Yasin. 2014. A top-down method for performance analysis and counters architecture. *Presented at the 2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. https://scholar.google.co.il/scholar?q=A+top-down+method+for+performance+analysis+and+counters+architecture.&btnG=&hl=en&as_sdt=0%2C5.

Received June 2016; revised October 2016; accepted November 2016