Empirical Study of Power Consumption of x86-64 Instruction Decoder

Mikael Hirki[1,2], Zhonghong Ou[3], Kashif N. Khan[1,2], Jukka K. Nurminen[1,2,4], and Tapio Niemi[1]

[1]*Helsinki Institute of Physics* [2]*Aalto University* [3]*Beijing University of Posts and Telecommunications* [4]*VTT Research*

Abstract

It has been a common myth that x86-64 processors suffer in terms of energy efficiency because of their complex instruction set. In this paper, we aim to investigate whether this myth holds true, and determine the power consumption of the instruction decoders of an x86-64 processor. To that end, we design a set of microbenchmarks that specifically trigger the instruction decoders by exceeding the capacity of the decoded instruction cache. We measure the power consumption of the processor package using a hardware-level energy metering model called the Running Average Power Limit (RAPL), which is supported in the latest Intel architectures. We leverage linear regression modeling to break down the power consumption of each processor component, including the instruction decoders. Through a comprehensive set of experiments, we demonstrate that the instruction decoders can consume between 3% and 10% of the package power when the capacity of the decoded instruction cache is exceeded. Overall, this is a somewhat limited amount of power compared with the other components in the processor core, e.g., the L2 cache. We hope our finding can shed light on the future optimization of processor architectures.

1 Introduction

Making data centers more energy efficient requires finding and eliminating new sources of inefficiency. With the recent introduction of ARM-based servers, we will soon have more CPU architectures to choose from. These servers are based on the new ARMv8 architecture that adds support for 64-bit computing. The first prototypes started shipping in 2014 [1]. Energy efficiency of the first models has been somewhat poor but efficiency is expected to improve in later models [2].

The most commonly cited difference between ARM and Intel processors is the instruction set [5]. Intel processors use the x86-64 instruction set, which is a Complex Instruction Set Computer (CISC) architecture; while ARM designs are based on the Reduced Instruction Set Computer (RISC) architecture. CISC instructions are more complex: a single instruction can load a value from memory and add it to a register. Today these CISC instructions are decoded into smaller micro-operations, i.e. one instruction is translated into one or more micro-operations. A common myth is that decoding the x86-64 instructions is somehow expensive.

In this paper, we aim to investigate the myth that decoding x86-64 instructions is expensive. We leverage the new features that are present in Intel processor architectures from Sandy Bridge onwards. The two features are the Running Average Power Limit (RAPL) and a micro-op (short for micro-operation) cache. RAPL [6, 8] allows measuring the processor energy consumption at high accuracy. The micro-op cache allows the instruction decoders to be shut down while decoded instructions are served from the cache, thus improving energy efficiency. Our core idea is to design microbenchmarks that exhaust the micro-op cache, thus triggering the instruction decoders. This allows us to specifically measure the power consumed by the instruction decoders. The code for our microbenchmarks is available at https://github.com/mhirki/idq-bench2.

Our paper makes the following major contributions:

- We develop a set of microbenchmarks to accurately measure the power consumption of the instruction decoders in an x86-64 processor.
- We show that the percentage of power consumed by the instruction decoders is between 3% and 10% of the total package power.
- We conclude that the x86-64 instruction set is not a major hindrance in producing an energy-efficient processor architecture.

The rest of the paper is structured as follows. Section 2 gives a more detailed explanation of processor architecture and the Intel RAPL feature. Section 5 reviews the related work. Section 3 presents the hardware and software configuration as well as the implementation of our microbenchmarks. Section 4 presents the results obtained using our microbenchmarks. Finally, Section 6 concludes our paper and presents an idea for future work.

2 Background

2.1 Processor Architecture

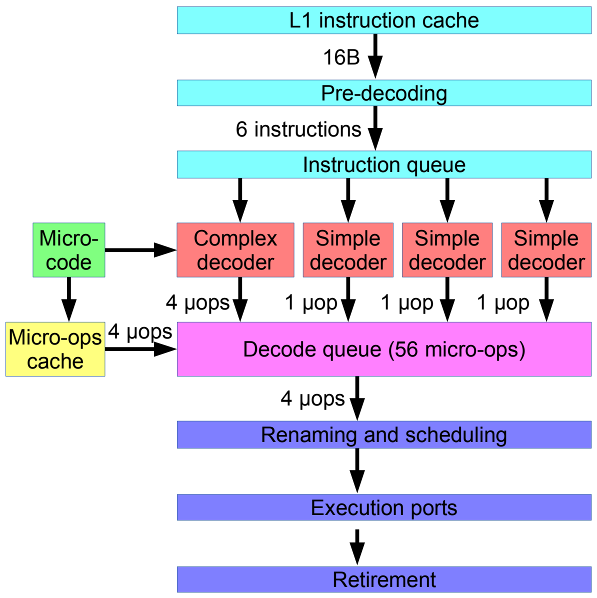

Figure 1: Simplified version of the Intel Haswell pipeline.

Figure 1 depicts a simplified version of the Intel Haswell pipeline. This simplified version highlights the micro-op cache, a new feature first introduced in the Sandy Bridge architecture. Before the new cache, the Intel architecture relied on a very small decoded instruction buffer and four instruction decoders. Three of the decoders are simple decoders that can only decode a single instruction into a single micro-op. The fourth decoder can receive any instruction and produce up to four micro-ops per cycle. Thus, instructions that are translated into more than one micro-op can only be decoded by the complex decoder. This creates a potential bottleneck if the frequency of complex instructions is high.

The micro-op cache solves performance problems in some cases. In addition, it allows shutting down the instruction decoders when they are not used, thus saving power. The capacity of the micro-op cache is 1536 micro-ops, which is equivalent to eight kilobytes of x86 code at maximum.

2.2 Running Average Power Limit (RAPL)

Intel introduced the RAPL feature in their Sandy Bridge architecture. It produces an estimate of the energy consumed by the processor using a model based on microarchitectural counters. RAPL is always enabled in supported models and the energy estimates are updated every millisecond. RAPL supports up to four different power domains depending on the processor model. The Package domain estimates the energy consumed by the entire processor chip. Power plane 0 is a subdomain which estimates the energy consumed by the processor cores. Power plane 1 is another subdomain which estimates the energy consumed by the integrated graphics in desktop models. The DRAM domain is a separate domain which estimates the energy used by the dual in-line memory modules (DIMMs) installed in the system. In this paper, we present measurements of the RAPL Package domain exclusively.

3 Methodology

3.1 Hardware and Software

Table 1: Hardware used in the experiments.

Processor:	Intel Core i7-4770 @ 3.40 GHz
Architecture:	Haswell
Cores:	4
L3 cache:	8 MB
Turbo Boost:	Supported but disabled
Hyperthreading:	Supported but disabled
RAM:	2x 8 GB Kingston DDR3, 1600 MHz (dual-channel)
Motherboard:	Intel DH87RL
Power supply:	Corsair TX750

The hardware used in the experiments is listed in Table 1. An Intel Haswell desktop processor, released in 2013, is used for the experiments. The Turbo Boost feature is disabled to make the experiments repeatable without relying on thermal conditions. Meanwhile, disabling Turbo Boost ensures that the processor does not exceed its base frequency of 3.4 GHz. In addition, the hyperthreading feature is disabled in the BIOS of the machine on purpose. This gives access to eight programmable performance counters, compared with only four when hyperthreading is enabled.

Table 2: Software and kernel parameters used in the experiments.

Operating system:	Scientific Linux 6.6
Kernel:	Linux 4.1.6
Cpufreq governor:	Performance
NMI watchdog:	Disabled
Transparent hugepages:	Disabled
Compiler:	GCC 4.4.7
Performance measurement:	PAPI library 5.4.1 and Perf 4.1.6

Table 2 states the software components used in the experiments. Certain kernel settings are adjusted to ensure that the experiments are repeatable. The Cpufreq governor is set to *performance* that runs the processor at the highest available frequency when it is not idle. This prevents the processor from running at frequencies lower than 3.4 GHz. The Non-maskable interrupt (NMI) watchdog is disabled, which frees up one programmable performance counter. In addition, support for transparent hugepages is disabled since it can change performance characteristics while programs are running. The performance counters are read using both the PAPI (Performance Application Programming Interface) library [14] and the Perf tool. Perf is a low-level performance analysis tool that is shipped with the Linux kernel. The RAPL counters are read using the MSR driver interface.

GCC 4.4.7 is used as the compiler in the experiments since it is the default for the Linux distribution. The same compiler is likely still used by many applications running on Scientific Linux 6. The compiler flag *-O2* is used to produce optimized binaries. Model-specific optimizations, such as Advanced Vector eXtensions (AVX), are not used.

3.2 Microbenchmark Design

We design a series of microbenchmarks to measure the power consumption of the instruction decoders of the x86-64 processor. The benchmarks execute a single line of code in a loop. We manually unroll the loop up to 2048 times. This allows us to easily increase the size of the code to exceed the capacity of Intel's micro-op cache. In turn, it forces the processor to use the instruction decoders at least periodically. We can then compare the power consumption against a loop with a smaller unroll count. If the loop is still executed in the same amount of time, the observed difference in power consumption can be attributed to the instruction decoding pipeline.

Loop unrolling is a well-known optimization technique used by compilers. In simple cases, where the number of iterations n is divisible by some integer k, the loop body can simply be repeated k times. The number of iterations for the unrolled loop is n/k. Unrolling increases performance by minimizing the number of branch instructions that need to be executed. Branch instructions are used for jumping from the end of the loop to its beginning. A correctly-predicted branch instruction consumes only one clock cycle in a pipelined processor. Thus, loop unrolling is most beneficial for very small loops.

The microbenchmarks use simple arithmetic operations such as addition and multiplication. Different benchmarks use either floating-point (double- or single-precision) or integer (32-bit or 64-bit) data types. They operate either on large arrays or simply on values stored in registers. The following code listing written in the C language shows the operations used in microbenchmark #1:

```
D += A[j] + B[j] * C[j];
```

Here A, B and C are floating-point arrays stored in memory. The variable D contains a floating point value stored in a register. The benchmarks do not write to memory because that would require an additional parameter in the power model. The following C code shows microbenchmark #2:

```
D += (A[j] << 3) * (A[j] << 4) *
     ((B[j] << 2) * 5 + 1);
```

In this case, A and B are integer arrays. Since instructions that access memory generate at least two micro-ops and hence are forced through the complex decoder, we need to add several filler operations that generate only one micro-op. Here we use bitwise shifts, multiplications and additions as filler operations. This approach eliminates the complex decoder as a performance bottleneck, allowing us to stress all four decoders.

We write many variations of each benchmark. In addition to the previous two benchmarks, we have dozens of different variations of them. We vary the type of operations (addition, multiplication, bitwise shift) used in the benchmarks. Furthermore, we change the size of the arrays so that they fit different caches such as L1 or L2 caches. We also vary the data types (double- or single-precision). The filler operations introduce additional opportunities for instruction-level parallelism, thus putting additional load on the instruction decoders.

4 Experimental Evaluation

4.1 Power Consumption versus Code Size

Our first step is to try a large number of different unroll counts using only a single benchmark. This gives us a better understanding of how energy consumption develops as the code size increases due to higher unroll count. We can then identify which components are responsible for the changes in energy consumption.

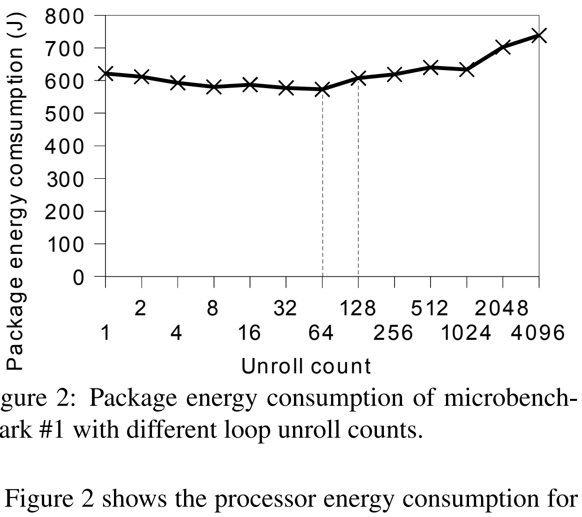

Figure 2: Package energy consumption of microbenchmark #1 with different loop unroll counts.

Figure 2 shows the processor energy consumption for different unroll counts. The energy is measured in joules as reported by the Intel RAPL feature. From the figure, we can see that energy consumption initially decreases until it reaches its minimum at unroll factor of 64. We do not observe any performance gains in the running time of the loop. Therefore, the reduction in energy consumption can be attributed to the reduced branch prediction activity. At unroll factor of 128, we can see an increase in energy consumption that is caused by the instruction decoders. After this, the energy consumptions grows as the hit ratio for the micro-op cache drops. Between unroll factors 1024 and 2048, we see a big jump in energy consumption that is due to the L1 instruction cache capacity being exceeded. We see that the energy consumption continues growing as the L2 cache activity increases.

Based on Figure 2, we can extrapolate that energy consumption will keep growing for an even larger loop as the L2 cache will be exhausted. Eventually, the translation lookaside buffer (TLB) will also be exhausted, at which point there will be a big performance penalty and correspondingly, big increase in energy consumption.

4.2 Regression Modeling

We employ linear regression modeling to further isolate the power consumption of the instruction decoders. We implement two cases with different unroll counts in each of our microbenchmarks. The extreme case is the point when the capacity of the micro-op cache is exceeded. From Figure 2, we can see that this point is at an unroll count of 128 for benchmark #1. This is our *extreme* case. We also define a *normal* case for comparison that is the extreme case's unroll count divided by two. In this example, the normal case is at an unroll count of 64, which is the energy minimum in Figure 2. Having dozens of benchmarks that implement these two cases allows us to produce enough data to accurately model the power consumption of different processor components such as the execution units, the instruction decoders and the different caches.

$$\begin{aligned} P_{\text{package}} = 6.05 &+ \frac{\text{cycles}}{\text{second}} \times 1.63 \times 10^{-9} \\ &+ \frac{\mu\text{ops issued}}{\text{second}} \times 2.15 \times 10^{-10} \\ &+ \frac{\mu\text{ops decoded}}{\text{second}} \times 1.40 \times 10^{-10} \\ &+ \frac{\text{L1 hits}}{\text{second}} \times 4.35 \times 10^{-10} \\ &+ \frac{\text{L2 references}}{\text{second}} \times 4.05 \times 10^{-9} \end{aligned} \quad (1)$$

Table 3 lists the performance events selected. They reflect the components stressed by our benchmarks. The total number of benchmarks we use is 49. This number includes the different variants of each benchmark. In addition, each benchmark has two cases: the normal one and the extreme one. We run each case for 11 seconds on our test machine. During the execution, we measure power consumption at a rate of 50 samples per second using RAPL. We also use the Perf tool to record performance events at the same rate. We need to adjust the Perf timestamps to match with our power consumption data because the Linux kernel does not use real time for performance events. In addition, we have to filter out anomalous values in the performance data. We use the ordinary least squares method to construct a linear model for the package power consumption.

Equation 1 shows the resulting power model obtained using linear regression. The result is the predicted package power consumption as measured in watts. The input parameters are measured as the number of events per second. For example, the value of cycles/second would be 3.4 billion (or 3.4×10^9) for a single core of our test machine. The coefficient of determination (R^2) for our model is 0.989. Therefore, we believe the model accurately represents the different CPU components when running our benchmarks.

4.3 Power Breakdowns

In Table 4, we select two microbenchmarks for more detailed examination. We select benchmark #1 because of its high L2 & L3 cache power consumption. The L2 and L3 caches are major components and they allow putting the instruction decoders into perspective. Benchmark #2 is selected because of the high power consumption in the instruction decoders. At the same time, #2 does not use the L2 cache. Therefore, #2 demonstrates the maximum power consumption of the instruction decoders.

We discovered that our benchmarks appear to trigger the L2 prefetchers even though the data fits into the L2 cache. This problem appears to be limited to synthetic benchmarks running on the Haswell platform. The prefetchers appear to be fetching data from the L3 cache, which causes additional power consumption. Therefore, we have labeled the component as "L2 & L3 cache" in our power breakdowns.

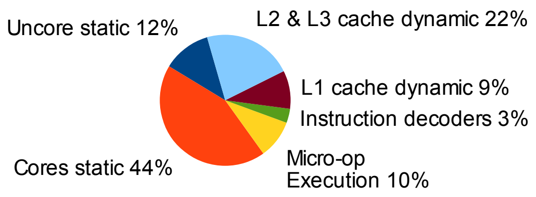

Figure 3: Package power consumption breakdown of microbenchmark #1, which stresses the L2 cache using floating-point operations.

Figure 3 shows the power consumption breakdown for our microbenchmark #1. It demonstrates the power consumption of different CPU components as predicted by

Table 3: Performance events used in the linear regression model.

Event name:	Description:
CPU_CLK_UNHALTED.THREAD_P	The number of clock cycles for each core.
UOPS_ISSUED.ANY	The number of micro-ops issued to the execution units.
IDQ.MITE_UOPS	The number of micro-ops produced by the instruction decoders.
MEM_LOAD_UOPS_RETIRED.L1_HIT	The number of hits in the L1 data cache.
L2_RQSTS.REFERENCES	The number requests to the L2 cache (including L2 prefetchers).

Table 4: Power consumption breakdown by processor components for two different microbenchmarks as predicted by our model.

Microbenchmark:	#1	#2
Type:	Floating-point	Integer
Instructions per cycle:	1.67	3.86
Uncore (W):	6.0	6.0
Cores (W):	22.1	22.1
Execution units (W):	4.9	10.4
Instruction decoders (W):	1.8	4.8
L1 cache (W):	4.8	3.8
L2 & L3 cache (W):	11.2	0.1
Micro-op cache hit ratio:	44.8%	29.6%

our power model. This figure corresponds to the benchmark #1 in Table 4. We have labeled the constant term as *uncore static* in this figure since it roughly corresponds to the static power consumption of the uncore component in the processor. This term includes the static power consumption of components such as the L3 cache and the memory controller.

Based on Figure 3, the power consumption of the instruction decoders is very small compared with the other components. Only 3% of the total package power is consumed by the instruction decoding pipeline in this case. It should be noted that the hit ratio for the micro-op cache is 45%, which means that only 55% of micro-ops come from the decoders. Therefore, the instruction decoders can in theory consume twice as much power. This can happen with older generation architectures like Nehalem, which is the predecessor of Sandy Bridge. The Nehalem architecture lacks the micro-op cache, so it has to decode every single instruction in our benchmark.

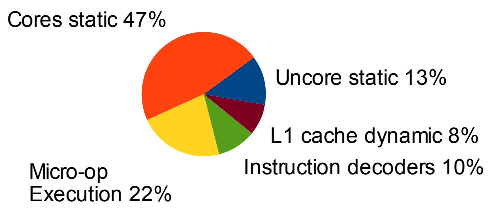

Figure 4: Power consumption breakdown of microbenchmark #2, which stresses the L1 cache using integer operations.

Figure 4 illustrates a power breakdown for our benchmark #2, which uses integer operations (as opposed to floating-point operations used in benchmark #1). Because of this, benchmark #2 reaches a much higher instructions per cycle (IPC) count. The higher IPC count causes the execution units and instruction decoders to consume more power. The micro-op cache hit ratio for #2 is also lower, which causes even more work for the instruction decoders. As a result, the instruction decoders end up consuming 10% of the total package power in benchmark #2. Nevertheless, we would like to point out that this benchmark is completely synthetic. Real applications typically do not reach IPC counts as high as this. Thus, the power consumption of the instruction decoders is likely less than 10% for real applications.

5 Related Work

Many power models have been proposed for modeling the power consumption of entire computer systems and individual components [4, 7, 13, 15]. McCullough et al. [13] evaluated earlier approaches to modeling power consumption. Their conclusion is that quadratic and other non-linear models are often required for accurate power modeling. This is due to the fact that many utilization metrics or performance counters do not scale linearly with low-level hardware activity. In this paper, we use a linear model because we choose components that can be modeled linearly using the parameters available. In addition, we use the Haswell architecture, which is significantly more energy efficient than older architectures. Our selection of performance events is also different from existing work. For example, we use the IDQ.MITE_UOPS performance event introduced in Sandy Bridge. To our best knowledge, Oboril et al. [15] are the only ones who have used this performance event besides us. They present power consumption breakdowns similar to ours. However, they do not use microbenchmarks specifically targeting the instruction decoders like we do.

The energy efficiency of different Intel and ARM platforms has been compared using numerous workloads [2, 3, 11, 16]. The results suggest that ARM gives good energy efficiency in specific workloads using specific parameters while Intel has more stable performance across a wider range of parameters. In addition, a few studies have attempted to investigate the significance of the instruction set [5, 10]. Blem et al. [5] concluded that the instruction set by itself is a fairly insignificant part of whole microprocessor design. A limitation of their work is that they did not measure the instruction decoders directly. Our paper addresses this limitation. Isen et al. [10] stated that Intel benefits from advanced microarchitectural features such as micro-op fusion that has helped close the gap between CISC and RISC instruction sets.

Microbenchmarks have been used for deriving power models before. Isci and Martonosi [9] use microbenchmarks to derive the power consumption of 22 different components of the Pentium 4 processor. Similar to our work, they use performance counters to estimate the activity of different components. However, our microbenchmarks are more focused on the instruction decoders. Leng et al. [12] model the power consumption General Purpose GPUs (GPGPUs). Their work shows that a microbenchmark-based approach works even for a different class of hardware.

6 Conclusion and Future Work

We designed a series of microbenchmarks to determine the power consumption of the instruction decoders in our x86-64 processor. We model the power consumption using linear regression analysis. Our linear model predicts the power consumption of different components including the execution units, instruction decoders, and L1 and L2 caches. The result demonstrates that the decoders consume between 3% and 10% of the total processor package power in our benchmarks. The power consumed by the decoders is small compared with other components such as the L2 cache, which consumed 22% of package power in benchmark #1. We conclude that switching to a different instruction set would save only a small amount of power since the instruction decoder cannot be eliminated completely in modern processors.

In the future, we plan to port our microbenchmarks to an ARM platform. This allows us to directly compare the energy consumption of different components between two different architectures. Another interesting platform to benchmark would be the Intel Atom, which is a low-power x86 design. The Atom does not support AVX instructions, which should simplify the decoders and thus reduce their power consumption.

7 Acknowledgments

This study was supported by the Green Big Data project of the Helsinki Institute of Physics. We would like to thank David Abdurachmanov, Peter Elmer and Giulio Eulisse for their ideas and feedback.

References

[1] AppliedMicro announces the availability of X-C1 development kits featuring X-Gene, Nov. 2014. (press release).

[2] ABDURACHMANOV, D., BOCKELMAN, B., ELMER, P., EULISSE, G., KNIGHT, R., AND MUZAFFAR, S. Heterogeneous high throughput scientific computing with APM X-Gene and Intel Xeon Phi. In *Journal of Physics: Conference Series* (2015), vol. 608, IOP Publishing.

[3] AROCA, R. V., AND GONÇALVES, L. M. G. Towards green data centers: A comparison of x86 and ARM architectures power efficiency. *Journal of Parallel and Distributed Computing 72*, 12 (2012), 1770–1780.

[4] BIRCHER, W. L., AND JOHN, L. K. Complete system power estimation using processor performance events. *IEEE Transactions on Computers 61*, 4 (2012), 563–577.

[5] BLEM, E., MENON, J., VIJAYARAGHAVAN, T., AND SANKARALINGAM, K. ISA wars: Understanding the relevance of ISA being RISC or CISC to performance, power, and energy on modern architectures. *ACM Trans. Comput. Syst. 33*, 1 (Mar. 2015), 3:1–3:34.

[6] DAVID, H., GORBATOV, E., HANEBUTTE, U. R., KHANNA, R., AND LE, C. RAPL: Memory power estimation and capping. In *Proceedings of the 16th ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED)* (2010).

[7] ECONOMOU, D., RIVOIRE, S., KOZYRAKIS, C., AND RANGANATHAN, P. Full-system power analysis and modeling for server environments. In *Proceedings of the Workshop on Modeling, Benchmarking, and Simulation (MoBS)* (June 2006).

[8] HÄHNEL, M., DÖBEL, B., VÖLP, M., AND HÄRTIG, H. Measuring energy consumption for short code paths using RAPL. *ACM SIGMETRICS Performance Evaluation Review 40*, 3 (2012), 13–17.

[9] ISCI, C., AND MARTONOSI, M. Runtime power monitoring in high-end processors: Methodology and empirical data. In *Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture* (2003).

[10] ISEN, C., JOHN, L. K., AND JOHN, E. A tale of two processors: Revisiting the RISC-CISC debate. In *Proceedings of the 2009 SPEC Benchmark Workshop on Computer Performance Evaluation and Benchmarking* (2009), Springer, pp. 57–76.

[11] JARUS, M., VARRETTE, S., OLEKSIAK, A., AND BOUVRY, P. Performance evaluation and energy efficiency of high-density HPC platforms based on Intel, AMD and ARM processors. In *Energy Efficiency in Large Scale Distributed Systems*. Springer, 2013, pp. 182–200.

[12] LENG, J., HETHERINGTON, T., ELTANTAWY, A., GILANI, S., KIM, N. S., AAMODT, T. M., AND REDDI, V. J. GPUWattch: enabling energy optimizations in GPGPUs. *ACM SIGARCH Computer Architecture News 41*, 3 (2013), 487–498.

[13] MCCULLOUGH, J. C., AGARWAL, Y., CHANDRASHEKAR, J., KUPPUSWAMY, S., SNOEREN, A. C., AND GUPTA, R. K. Evaluating the effectiveness of model-based power characterization. In *Proceedings of the USENIX Annual Technical Conference (ATC)* (2011).

[14] MUCCI, P. J., BROWNE, S., DEANE, C., AND HO, G. PAPI: A portable interface to hardware performance counters. In *Proceedings of the Department of Defense HPCMP Users Group Conference* (1999), pp. 7–10.

[15] OBORIL, F., EWERT, J., AND TAHOORI, M. B. High-resolution online power monitoring for modern microprocessors. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)* (2015), EDA Consortium.

[16] OU, Z., PANG, B., DENG, Y., NURMINEN, J. K., YLA-JAASKI, A., AND HUI, P. Energy- and cost-efficiency analysis of ARM-based clusters. In *Proceedings of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)* (2012).