Article

An Accurate Estimation of the Energy Cost of Dynamic Branch Prediction in an Intel High-Performance Processor

Fahad Alqurashi [1,*] and Muhammad Al-Hashimi [1]

[1] Department of Computer Science, Faculty of Computing and Information Technology, King Abdulaziz University, Jeddah 25732, Saudi Arabia; mhashimi@kau.edu.sa (M.A.-H.)
* Correspondence: falqurashi0041@stu.kau.edu.sa

Abstract: Power and energy efficiency are among the most crucial requirements in high-performance and other computing platforms. This work examines through extensive experimentation methods and procedures suitable for assessing the power and energy efficiency of fundamental hardware building blocks inside a typical high-performance CPU, focusing on the dynamic branch predictor (DBP). The investigation relies on the Running Average Power Limit (RAPL) interface from Intel, a software tool for credibly reporting the power and energy based on instrumentation inside the CPU. We use well-known microbenchmarks under various run conditions to explore potential pitfalls and to develop precautions to raise the precision of the measurements obtained from RAPL for more reliable power estimation. The authors discuss the factors that affect measurements and share the difficulties encountered and the lessons learned.

Keywords: Dynamic Branch Predictor, Power Efficiency, Energy, HPC, RAPL, microbenchmarks

1. Introduction and Motivation

In the past, the primary interest of computer architects and software developers was to increase performance. But in the last couple of decades, power and energy efficiency emerged as major requirements in computing. One of the reasons that led to power and energy efficiency interest in computing is the need to reduce the power consumption of HPC to achieve exascale supercomputers [1]. This goal motivated researchers to investigate the fundamental software building blocks such as sort algorithms, matrix multiplication algorithms, and shortest path algorithms used commonly in HPC applications to find the factors that notably affect power consumption. Similarly, several fundamental hardware components, such as arithmetic units, decoders, caches, and Dynamic Branch Predictors (DBP), may be of interest in terms of power efficiency.

This work focuses on investigating DBP for various reasons. Firstly, DBP became an essential component in all modern CPUs used in HPC. Secondly, conditional jump instructions represent a significant percentage of most typical application instructions. Modern CPUs use DBP to guess the direction and the target address of jump instructions which means a heavy utilization of DBP during any application run. Consequently, any power savings related to this component may yield substantial benefits in the total energy consumption. Thirdly, according to some statistics, DBPs account for 10 to 40 percent of CPU dynamic power consumption [2]. Fourthly, the recent DBP security issues appeared, and the solutions proposed to mitigate them need investigation from a power and energy perspective [3–5]. Lastly, the lack of research papers studying DBP and its security issues from a power and energy perspective gave us an extra boost for investigation.

As many research papers pointed out, software style impacted program performance and power and energy consumption of computing devices in significant ways [6]. The impact demanded that developers find easy ways to measure performance and power consumption of their software. RAPL, introduced in modern processors, is one of the most prominent software tools that can report the power in different CPU domains [7]. RAPL

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 9 June 2023 doi:10.20944/preprints202306.0725.v1

obviates the need for external hardware power measurement devices that are complex to manage, lack fine granularity, and are expensive [8]. However, using RAPL for measuring CPU power consumption accurately, especially on the process level, needs some precautions to avoid noise in measurement [8].

This report presents the researchers' experience and methodology in using the RAPL tool for measuring the power consumption of a program and the precautions recommended during measurement operations to raise the reliability of the RAPL tool for more accurate power estimation. The paper is organized as follows. Section 2 is a literature review of previous works that evaluated the power and energy consumption of fundamental software building blocks. Section 3 is a background about DBP and RAPL. Section 4 explores various factors that affect the accuracy of reading results of RAPL that act as source of noise. Section 5 examines the different ways to acquire more reliable results from RAPL tools available in Linux. Section 6 focuses on the methodology used to estimate the power and energy consumed by the DBP. Section 7 introduces experimentation results that motivate the precautions mentioned in Section 5. Section 8 presents some conclusions.

2. Literature Review

Researchers have recognized the impact of software on computing power and energy consumption since the 1990s. Mehtal *et al.* [9] proposed software techniques to reduce the energy consumption of a processor using compiler optimization, such as loop unrolling and recursion elimination. They also studied how various algorithms and data structures affect power and energy usage.

Capra and Francalanci [10] considered the design factors that influence the energy consumption of applications that perform the same functionality but have different designs. They experimentally assessed the energy efficiency of management information systems. In their study, they found that application design has a significant impact on energy consumption.

Sushko *et al.* [11] studied the effect of loop optimization on the power consumption of portable computing devices. They applied their study to ARMv8 architectures. Their study showed the power efficiency gained by fitting data on cache and parallelization for loop optimization.

Al-Hashimi *et al.* [12] studied the effects on the system power consumption of three iteration statements: For-loop, While-loop, and Do-While. They measured for each case the average time, power, temperature, the value of the maximum temperature, and the number and percentage of times reached. They found that the For-loop was the most power efficient and that the While Loop had the worst power efficiency.

Abulnaja *et al.* [13] analyzed bitonic mergesort compared to an advanced quicksort on the NVIDIA K40 GPU for power and energy efficiency. They introduced the factors that affected power consumption and studied those that lead to higher energy and power consumption, such as data movement and access rate. They concluded that bitonic mergesort is inherently more suitable for the parallel architecture of the GPU. This study triggered the investigation of more software building blocks, such as spanning tree algorithms and binary search algorithms.

Aljabri *et al.* [14] conducted a comprehensive empirical investigation into the power efficiency of mergesort compared to a high-performance quicksort on the Intel Xeon CPU E5-2680 (Haswell), which is more commonly used in HPC and has more accurate sensor readings than previous generation Intel Xeon E5-2640 CPU (Sandy Bridge) utilized in an earlier work [15]. The research was motivated by the fact that divisions by powers of two, the most frequent operation in mergesort, may be performed by a power-efficient barrel shifter. Mergesort applies in its procedure a divide-and-conquer strategy in which the original list (or array) is divided, recursively, into two equal lists. The study concluded that mergesort had an advantage from the power efficiency side on quicksort, with comparable time efficiency of the two algorithms. This study encouraged more investigation into

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 9 June 2023 doi:10.20944/preprints202306.0725.v1

other algorithms that perform similar tasks but have different time efficiency from a power perspective.

NZ Oo *et al.* [16] studied Strassen's matrix multiplication algorithm from a performance vs. power perspective. In their study, they found a way to enhance performance and reduce energy consumption by using loop unrolling on the recursive level of the algorithm to minimize the cache misses and increase the data locality. They claim that their method increased performance by 93 percent and reduced energy consumption by 95 percent.

Jammal *et al.* [17] studied the power efficiency of three matrix multiplication algorithms: a definition-based, Strassen's divide-and-conquer, and an improved divide-and-conquer on the Intel Xeon CPU E5-2680. The main finding of this work is that the fastest divide and conquer algorithm is power-efficient only for small matrix sizes. For larger sizes, the definition-based algorithm turned out to be more power-efficient. They also studied the effect of every cache level miss on power consumption.

Some of the previous works hypothesized that some algorithms have superiority over others that are equivalent in tasks in terms of power efficiency, but others have an advantage in terms of time efficiency. Those studies concerned the factors that played significant roles in power saving to open the door for further investigation that can lead to a balance between power and time efficiency.

In this work, the researchers were motivated by the previous work to investigate fundamental hardware building blocks inside the CPU rather than software building blocks from a power and energy perspective. A recent study by Lastovetsky *et al.* [18] introduced methods usable as building blocks to scale HPC computing systems to achieve energy and performance optimization on the application level. They required energy profiling of computational components. They introduced some precautions to reduce the noise in the measurements. While their study considered the whole system, this study was concerned with individual CPU components, specifically the dynamic branch predictor.

3. Background

This section introduces a background of dynamic branch prediction and the RAPL interface.

3.1. Dynamic Branch Predictor

One of the most efficient features used inside modern CPUs to attain a high performance by increasing Instruction Level Parallelism (ILP) is speculative execution. This feature overcomes stalling in some of the CPU's processing stages caused by structure, control, or data hazards. Branch instructions cause a main source for such dependencies. For instance, in conditional branch instructions, the CPU needs to decode and execute a branch instruction before it decides whether to take the branch by jumping to the branch instruction target address or continuing to the next instruction in the program sequence [19], which causes a control hazard. According to most program statistics, the branch instructions represent 10 to 20 percent of all program instructions in most workloads. This percentage warrants careful treatment by a CPU. If not, it would dramatically degrade the performance. Modern CPUs use speculative execution to predict the outcome of the branch instructions before decoding them.

In general, branch prediction techniques fall into static or dynamic. Static branch prediction mechanisms append a bit during program compilation to every branch instruction operation code (OPCODE), which indicates whether the branch is taken or not taken by assigning a zero or one to indicate taken or not taken, respectively. The criteria that static decision depends on is the nature of the branch instruction. For instance, unconditional branch and backward conditional instructions are always guessed as (taken), while forward conditional instructions are guessed as (not-taken) [1].

In contrast, in dynamic branch prediction (DBP) techniques, the predictor collects information about a branch instruction while running a program. This information is a sort of history that describes the behavior of the branch instruction during its last number of

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 9 June 2023 doi:10.20944/preprints202306.0725.v1

executions (taken or not taken and its target address). This information is used the next time the branch instruction is executed to guess its direction (taken or not taken.) Dynamic prediction is much more accurate than static prediction [20].

DBP is a principal component of speculative execution resources [21]. It is a digital circuit called the Dynamic Branch Unit (DPU), used in pipelined CPUs to guess the direction and the target address of the branch instructions to improve the flow in the pipeline. DBP has a significant positive impact on CPU performance. DBP is accessed almost every cycle on average, i.e., it consumes more power and dissipates more heat. So, any improvement in its power consumption leads to significant CPU power and energy efficiency [20,21].

Various DBP techniques emerged in the last three decades. Examples are two-level predictions, interference-reducing predictors, neural predictors, and hybrid branch predictors. In the earlier era of DBP design, the primary goal of these techniques is to improve the CPU's performance, which requires a balance between the accuracy of DBP prediction and the access time to it. However, power and energy awareness became a crucial goal of any modern CPU component, including DBP. So modern CPUs require trade-offs in design between cost, performance, and power consumption [20].

Even though the exact organization of a CPU's DBP usually is not made public, the industrial implementations of DBPs consist in general of five major parts that distinctly influence power consumption [22].

1. Branch Target Buffer (BTB): is a set-associative cache that stores the target addresses of conditional and unconditional branch instructions.
2. Indirect Branch target buffer (IBTB): is a direct-mapped cache that stores the target addresses of indirect branch instructions.
3. Loop predictor is a set-associative cache to predict the outcome of conditional branch instructions with loop behavior.
4. Global predictor is a set-associative cache to predict the outcome of the general conditional branch instructions.
5. Bimodal predictor: a two-bit saturating counter.

In this research, the authors hypothesize that just as there are factors that affect power consumption in some algorithms, there are factors that affect power consumption in DBP. The research interest of this work was to investigate these factors and try to stress the different DBP components to find out the components that may contribute to power consumption more than others.

3.2. Intel RAPL

RAPL (Running Average Power Limit), a feature of Intel CPUs since the advent of Sandy Bridge architecture, performs two tasks. The first and foremost task is limiting the energy consumption of different components in the CPUs to protect them from the thermal effect. The second task is to provide a software tool to measure the power and energy consumed by those different components, during the program run, in fine granularity [23,24].

RAPL is a valuable feature introduced, first, by Intel in the Sandy Bridge architecture in 2011 to help researchers and system designers to gain estimation for power and energy of different domains inside and outside CPU; namely, a whole package, core, un-core and DRAM [25]. Several free tools can benefit from the RAPL interface. Some of these tools can run under Windows and Mac operating systems, and others run under the Linux operating system.

In modern Intel CPUs, RAPL can limit and measure the power and energy of different levels (or domains) in the CPU. The biggest domain is called the package domain. It is where RAPL can control and measure the power and energy consumed by the whole CPU socket. The second domain, Power Plane 0 (PP0), deals with the total power/energy consumed by all cores in the CPU. The third one, called the DRAM domain, can measure the power and energy consumed by the dynamic RAM. The fourth domain, Power Plane 1 (PP1), can measure the power and energy consumed by GPU. Another one, called PSys,

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 9 June 2023 doi:10.20944/preprints202306.0725.v1

was introduced in some Intel CPUs to control and monitor the power and thermal impact of the previous domains in addition to eDRAM (the embedded DRAM integrated into the same CPU between cache level 3 and DRAM) and some other features in recent Intel CPUs, such as CPU System Agent responsible for handling I/O between the components and the CPU. However, not all Intel CPUs support all mentioned domains. One needs to review documents to check the power domains a CPU supports [25].

Several research papers investigated CPU power and energy consumption that confirmed the effectiveness of the RAPL tool. Giardino *et al.* used RAPL registers to calculate the average power for SPEC CPU2006 benchmarks through `perf_events` command in Linux [26]. Khan *et al.* demonstrated the accuracy of RAPL in power and energy estimation. They also showed some weaknesses and limitations of RAPL [8]. Desrochers et al. found that power and energy estimation for DRAM using RAPL matches the power and energy estimated by WattsUpPro, a power measurement device, with a constant offset between RAPL and WattsUpPro [23]. Zhang and Hoffmann evaluated RAPL as a power limit control. They concluded that RAPL achieves good power stability in stable applications running for a long time [27].

Khan *et al.* compared the power consumed by the CPU package, as measured by an external power measurement device connected to the wall socket, to the power consumption measured by RAPL. Their results demonstrated a strong correlation between the measurement obtained from the wall socket and that obtained from RAPL. Khan and Nizam introduced a comprehensive study about the accuracy of RAPL measurement. They concluded that the RAPL was effective and a suitable alternative to external hardware devices [28].

This work focuses on RAPL to measure the power and energy consumption of different CPU domains in servers, workstations, desktops, and laptops, as well as how to obtain more accurate and, hence, reliable readings.

Before RAPL, CPUs predicted and estimated power based on a group of performance counters that can model predictive estimations of power and energy. RAPL, on the other hand, is an onboard digital meter (a set of counters) that gives better estimates of the power and energy compared to the old way that depended only on modeling since it also relies on embedded voltage regulators. RAPL exposes readings to the software layer through a group of registers classified as Machine Specific Registers (MSRs). Reading from or writing to these registers can be done through the two privileged machine instructions: RDMSR and WRMSR. The RAPL MSRs are updated approximately every millisecond in Intel CPUs. So, if any MSR multiple reading operations happened during one millisecond, the read values would be the same and considered as old values [27].

4. Noise Sources in RAPL Power Estimations

The term *noise*, in the context of this work, is used in a specific way. It refers to any one of the many factors that can affect the accuracy of the intended measurements. It stems cheifly from the experimental environment but can also come from the methods and the tools behind the readings. Understanding the sources of noise can help obtain accurate readings for the intended measurement. This section elaborates on these factors.

4.1. CPU Temperature

The most important environmental factor is the CPU temperature. It is well known that there is a strong correlation between CPU temperature and RAPL readings [8]. It makes the surrounding temperature one of the most important causes of noise. The rise in CPU temperature could happen because of the lab temperature, the heavy utilization of the CPU, poor ventilation inside the computer case, or a faulty CPU heat dissipation system [14].

Therefore, the experiment must be conducted in a good air-conditioned environment to maintain a reasonable and consistent ambient temperature. Otherwise, RAPL readings

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 9 June 2023 doi:10.20944/preprints202306.0725.v1

will be affected by the ambient CPU temperature [14]. Reducing the effect of CPU-heavy utilization will be discussed in a later sub-section.

4.2. Cross Core Thermal Exchange Effect

The rise of temperature in a core can significantly influence its neighbor cores. This effect arises from the thermal exchange between adjacent physical cores in the same die. The heat seeps from a core to a neighboring one and may become a source of RAPL reading noise [14].

4.3. Using Multiple Cores for One Application

Some applications utilize multiple cores by design to leverage their performance in parallel. However, power estimation of applications that run on many cores leads to inaccurate results when using RAPL hardware counters. With such applications, we can set the affinity to restrict them to running on one core [14].

4.4. Context Switching between Applications and Operating System

Most applications need services from the operating system, such as file management and input/output device control. These services are not a part of the program intended to estimate its energy. So, we need to deal with these services carefully to ensure pure measurements of the power consumption due to the application. However, some researchers may consider these services a part of the application to include in a proper estimate of an overall power consumption [14].

4.5. Hyper-Threading Technology

Unlike multithreaded applications, where an application can run on different cores, the hyper-threading feature on some modern CPUs allows threads from many programs to run on the same processor core. Hence, one physical core can function as two logical cores. Hyper-threading feature can be enabled or disabled (enabled by default) by the user from the BIOS. For accurate power estimation, hyper-threading should be turned off to ensure that no other application than the one we intend to measure shares the same physical core [25].

4.6. Operating System Issues

When we use the RAPL interface to measure the power consumed by software, the operating system plays a significant role in the accuracy of results. The effect of the operating system on the readings comes in several manifestations.

Many functions, managed by the operating system, run in the background and contribute to the inaccuracy of power estimation. These functions should be isolated from the application one way or another [14].

When an application deals with a lot of data from the secondary storage, the loading operation consumes a large amount of power which is not due to the actual application logic. For accurate results, in this case, the data could be embedded in the application file to be stored in the main memory when the application loads [14].

Another feature controlled by the operating system is the power management feature in all operating systems today. The technology tries to implement the most efficient power mode as long as possible. For an accurate comparison of power and energy between two codes, this feature needs to be disabled [29].

4.7. Compiler Optimization Issues

Compilers come with many optimization options. These options may affect RAPL readings. For instance, loop unwinding increases program speed by reducing the number of loop windings. If we estimate the power for an application compiled with a loop unwinding optimization, the measured power will be for a modified code that may not match the original logic [30].

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 9 June 2023 doi:10.20944/preprints202306.0725.v1

Compiler power consumption optimization is another feature supported by popular compilers. When we try to estimate the power consumption of a code apart from the architecture or platform, it is more accurate to avoid such optimization to measure the native attributes of the code [31]. However, when the optimization exploits features specific to the architecture or platform and the code targets that environment, then it would be reasonable to apply that optimization. In general, for accurate power measurement, power optimization options should be turned on or off depending on the purpose of measurement, whether it targets the code itself or the platform, or both.

5. Accurate RAPL Power and Energy Estimation in Linux

In this section, the authors introduce RAPL interface tools available in Linux to read the MSR registers. They also discuss policies and procedures to get more accurate results.

5.1. Linux RAPL Tools

Several tools are available for Linux that can deal with the RAPL interface. The following only lists the most common ones.

TurboStat tool is a part of the Linux kernel (needs root privilege) that can read RAPL information from MSRs registers and gives information about the power consumption. It comes with many options that control the display of the information and the period time of information collection [32].

PowerTOP is another Linux tool that can estimate the power consumption in CPU, GPU, or DRAM. It has several options that define the domain of measurement and the way the information is displayed [33].

The `perf` tool is one of the easiest Linux power profilers. Using it, one can collect power from various CPU domains (package, core, GPU, Psys). `Perf` can be used as a primary interface to the kernel to report a set of RAPL counters [34].

5.2. Containerization Technology

Containers were originally Linux features. So, the majority of them are Linux-based. Containerization is a way to isolate the application by creating a run environment around it called a container. All application dependencies, such as libraries, configuration files, and binaries, are encapsulated with the application in a standardized lightweight environment called a container. This operation helps the applications behave consistently when run on different hosts or multiple times on the same host. To assess an application on different hosts, it is better to containerize it to compare the effect of different architectures or platforms on power consumption. This measure is perhaps also useful when comparing the power consumption or performance of algorithms or applications on the same machine. In this research, we will test, experimentally, the effect of containerization on the RAPL measurements.

Many engines, such as Docker, CoreOS RKT, and runC are available to containerize any application [35]. The popular Docker container will be used for experimentation.

5.3. CPU Affinity Setting

In multicore CPUs, the applications running can use, by default, any set of available cores. To restrict an application to specific cores, one should set the processor affinity to the intended cores. For a more accurate power estimation of an application, it should be set to run on one core. The measurement accuracy is due to the fact that restricting the application to one core minimizes the context switching and CPU migration, eliminates hyper-threading, and reduces cache misses. The rise in those factors represents major sources of noise of RAPL readings. The ***taskset*** command is used to set the processor affinity in Linux [36].

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 9 June 2023 doi:10.20944/preprints202306.0725.v1

5.4. Dedicating One Core to An Application

Setting affinity for a CPU core ensures that the application runs on a single core, but it is not enough to ensure that no other programs share the application on the core. To assign a processor core exclusively to an application, the operating system must reserve the core from the beginning (during boot operation) to prevent the system scheduler from using it for any process. After that, the reserved core may be assigned to the application whose power consumption is under measurement. To reserve a core in Linux, the ***isolcpus=***⟨***CPU_ID***⟩ kernel parameter is added to the GRUB boot loader configuration file [37]. For instance, to reserve core number 5 in the CPU, one must add ***isolcpus=5*** to the GRUB file and reboot the system to activate the configuration. More than one core may be reserved by separating their numbers by commas [38].

5.5. Minimizing CPU Heavy Utilization Effect

For averaging a large amount of RAPL readings, researchers ran the application many times to take the average of the results. Although this operation can give more reliable results by minimizing the noise effect, it could become a source of thermal noise because of the heavy utilization of the CPU. To make the most of averaging and avoid the heavy utilization effect, the CPU can take a short rest time between runs. One can automate this operation in Linux by using loop commands in a script file and inserting a Linux sleep command at the end of every run loop to pause the next iteration for a while and give the CPU a chance to cool down [14].

5.6. CPU Power Management Feature

The power management feature in modern CPUs can affect the RAPL reading results. So, the authors strongly recommend disabling this feature to get more accurate power estimation for the aimed application. This feature can be disabled or enabled from the BIOS [14].

5.7. Minimizing Script Commands Effect

In the case of running the application repeatedly to take the average power, the commands in the script file used to automate this operation are not a part of the application code to be measured. One may ignore the effect of script commands if the application execution time is too long compared to the script code. However, this effect may become significant and a source of inaccurate results for applications with small code footprints. Repeating the application to form a long enough code can mitigate the scripting effects [24].

5.8. DBP Power Estimation Methodology

In contrast to the goal of the previous research papers mentioned in the literature to investigate the fundamental software building blocks, the interest of this research is to examine the Dynamic Branch Predictor (DBP) as a fundamental hardware component.

Since RAPL can directly measure the power consumed by the whole CPU cores (PP0 domain), a way to estimate the power consumed by the DBP alone is needed. DBP is accessed almost every cycle [20], so interest will focus on the dynamic power of DBP. As a simplification, the authors assume that the DBP dynamic power is caused only by the branch instructions, and any other DBP access contributes to the DBP static power. As interest is not the estimation of the whole power consumed by the DBP (Static and Dynamic power), we assume that the equation of PP0 domain power can be formulated as follow:

$$TotalCoresPower = DynamicDBPPower + RestCoresComponentPower, \quad (1)$$

Where: ***Total Cores Power*** represents the total power measured by RAPL for PP0 domain, ***Dynamic DBP Power*** is the dynamic power consumed by DBP and ***Rest Cores Component Power*** is the static and dynamic power consumption of the rest component in the PP0 domain including DBP static power.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 9 June 2023 doi:10.20944/preprints202306.0725.v1

Table 1. Benchmarks Specifications

Benchmark	Workload Type	Computational Characteristics
Blackscholes	Financial computation tasks	Computation intensity, data dependency, irregular memory access patterns
Ferret	Search application	Search intensity, intensive communication between processes, distance calculations, data retrieval
Raytrace	Interactions between rays and objects in a scene	Computational intensity, data locality, memory footprint
FFT	Fast Fourier transform application	Complex mathematical computations, single instruction multiple data (SIMD)

Table 2. Experimental Environment Specifications. Lubuntu is a lightweight version of Ubuntu.

Processor	Intel **Xeon E5-2680v3**; physical/logical cores: 12/24
Cache	L1 (data/instruction): 32/32 KB; L2 per core: 256 KB; L3 shared: 32 MB
Compiler	GCC 7.5.0
OS	Lubuntu 18.04.6 LTS; kernel: 4.15

To do so, a set of microbenchmarks that enable isolation of the DBP dynamic power consumption from the rest of the PP0 domain needs to be applied. Furthermore, the microbenchmarks must stress the main components of DBP to amplify the dynamic power consumed by each one. Then, linear regression modeling may be applied to the collected data to calculate the dynamic power consumed by the DBP and its major component. The appropriate branch-related events should be selected with matching microbenchmarks to build a suitable linear regression equation.

6. Experimentation

This section reports on the experimentation that examined the points discussed in Sections 4 and 5 about the factors that influence the accuracy of RAPL. In the experiments described in this section, four benchmarks of the PARSEC suite, described in Table 1, were used to test different aspects of the CPU's power. The PARSEC benchmark is an open-source suite comprised of a collection of parallel and multithreaded benchmark programs designed to simulate real-world computing scenarios. It includes applications such as fluid dynamics solvers, image processing algorithms, and data mining applications [39]. The suite is widely used to compare and evaluate different shared-memory architectures, programming models, and optimization techniques, including power modeling [40]. Table 2 describes the machine used for the experiments. The researchers opted for a lightweight version of Ubuntu to help further reduce OS-related environmental noise. Lubuntu uses the LXQt environment designed to be resource-efficient. It utilizes fewer system resources, such as CPU and RAM, which leads to lower power consumption. It also reduces system energy usage while applications run, making it a good choice for experimentation. The Linux `perf` tool collected data and CPU events for the four benchmarks. The numbers are detailed in Tables A1, A2, A3, and A4 in the appendix.

Each workload ran three hundred (300) times. Experimentation found that more runs did not contribute to the convergence of the average. The workloads ran with one thread (1T), two threads (2T), and four threads (4T). In each case, the benchmarks were executed inside Docker containers (IC) and outside (OC). The collected data included Execution Time (ET), Energy (E), Task Clock (TC), executed instructions (#inst) and Branches (#Br), Branch Misses (BrM), Last Level Cache-Store (LLCS), and -Load (LLCL), Last Level Cache Misses (LLCM), Context Switches (CS), CPU Migrations (CPUM), and Page Faults (PF). The researchers chose these events because of their direct impact on energy [40]. The data also were collected (outside containers) in two general cases: when the Power Management Unit was enabled (PMUen) and when the Power Management Unit was disabled (PMUdis).

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 9 June 2023 doi:10.20944/preprints202306.0725.v1

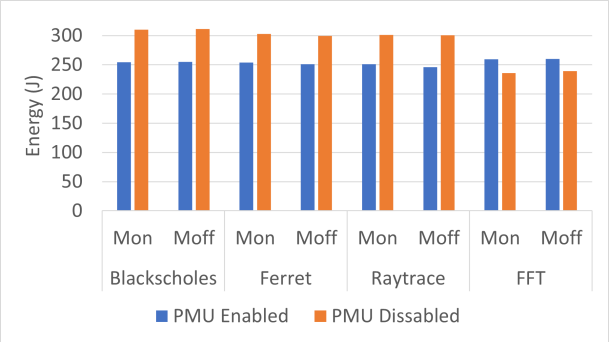


Figure 1. External Thermal Effect on RAPL readings.

To measure the effect of compiler optimization on RAPL measurements, the two benchmarks, Ferret and FFT, were compiled with the four main GCC compiler optimization level options, namely: O0, O1, O2, and O3, which range from minimal optimization (O0) to aggressive optimization (O3). The experiments also test the sensitivity of RAPL to the CPU ambient temperature by covering the upper side of the computer case with a mask with small holes. The rest of this section discusses, in detail, the results and the different scenarios used to obtain them.

6.1. Duscussion of Experimentation Results

6.1.1. External Thermal Effect

The authors attempted to measure the effect of the CPU ambient temperature to assess the sensitivity of the RAPL power measurement to external thermals. The test-bed computer case had ventilation holes on the rear and top sides. The execution time and the total energy of the four benchmarks were measured when the rear and top vents were open. They were measured again when the vents were masked. The two case conditions are labeled Mask-on (Mon) and Mask-off (Moff) in Table A2. Unexpectedly, the measurements were not very sensitive to ambient temperature in either disabled or enabled states of the PMU (see Figure 1). However, substantial changes in ambient temperature may affect the RAPL readings terribly.

6.1.2. Multithread Effect

When multiple threads are running on a processor, the processor has to work harder to execute the instructions of each thread, which can increase the instantaneous power consumption because more power is required to run the additional processing cores and to move data between them. However, running multiple threads may dramatically reduce energy consumption by allowing the processor to complete the task more quickly and enter a low-power state sooner.

To measure the effect of the number of threads on the total energy, average power, and the RAPL measurement accuracy, we run the benchmarks with 1, 2, and 4 threads outside containers (OC) and inside containers (IC). The results are summarized in Table 2 and shown in Figure 2.

There was a dramatic saving in total energy between 1 and 2 threads, ranging from 27.25% to 40.59% when running the benchmarks outside containers, while the savings in total energy when they ran inside container ranged from 28.35% to 54.00% with a slight increase in average power in two threads for both cases (OC & IC). The savings in total energy ranged from 23.74% to 39.78% between 2 and 4 threads outside containers, while the range of savings inside the containers was from 19.72% to 39.35%. See Figure 3.

6.1.3. Power Management Unit Effect

The Power Management Unit (PMU) is responsible for managing and controlling the power consumption of a computer system. It regulates various power-related aspects,

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 9 June 2023 doi:10.20944/preprints202306.0725.v1

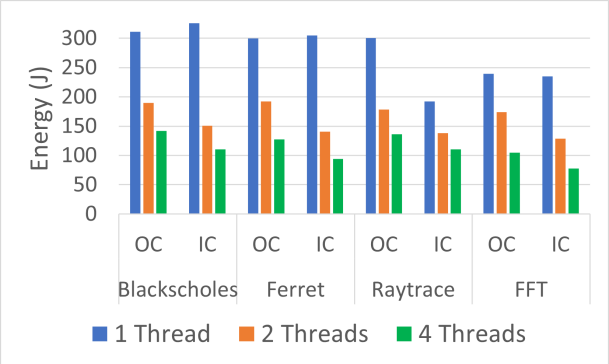


Figure 2. Multithreading Effect on Energy Consumption.

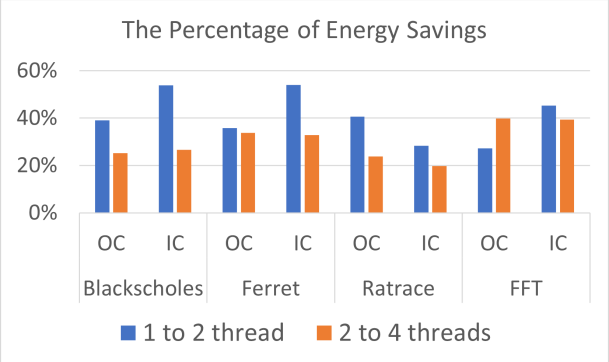


Figure 3. Effect of Number of Threads on Energy.

such as voltage levels, clock frequencies, and power states of different components. To measure the effectiveness of the Power Management Unit (PMU) on RAPL reading, we took the RAPL readings in two cases, Power Management Unit enabled (PMUen) and Power Management Unit disabled (PMUdis) as indicated in Table A1 and A2. Figure 4 shows an increase in execution time and energy and a decrease in average power in the case of PMUdis compared to the PMUen case in three of the benchmarks. However, in the case of the FFT benchmark, the energy consumed by the CPU in the case of PMUdis is less than the energy consumed in the case of PMUen.

6.1.4. Compiler Optimization Effect

Table A3 shows that compiler optimization dramatically affected the total energy and other program events. Not always, as one might expect, the more optimization, the better time and energy savings. In the Ferret benchmark, O1 optimization saved more power and energy than O2, while in FFT, O2 gave the best savings among the other optimization

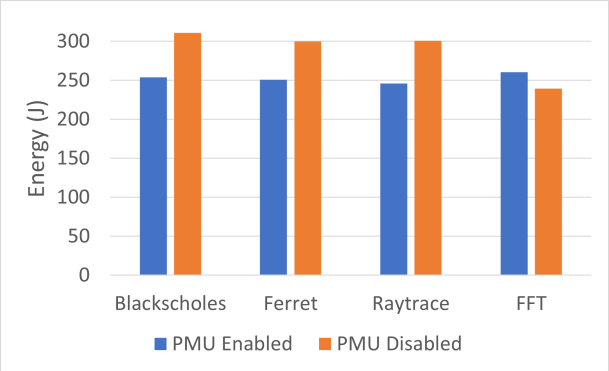


Figure 4. The Effect of PMU on Energy RAPL Reading.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 9 June 2023 doi:10.20944/preprints202306.0725.v1

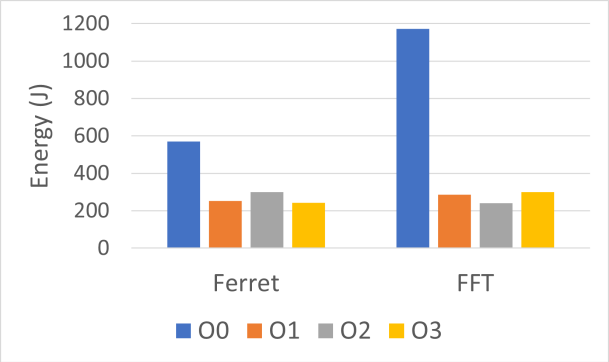


Figure 5. Effects of Compiler Optimization.

levels. O0 optimization (minimal optimization) always yields the highest energy readings (see Figure 5). However, it is safer to avoid compiler optimization when comparing two software codes from a power and energy perspective.

6.1.5. Containerization Effect

The benchmarks ran directly on the host and also ran again inside docker containers. These two cases are marked in Table A1 by OC (Outside Container) and IC (Inside Container). In most cases, running workloads inside the container gave a saving in the total energy compared to running them outside containers, even though the execution time was longer inside containers in most of the cases as expected. However, the case of 1 thread in Blackscoles and Ferret benchmarks gave more energy consumption inside the container. See Figure 6.

Running the benchmarks inside containers showed a slight increase in events compared to running them outside containers.

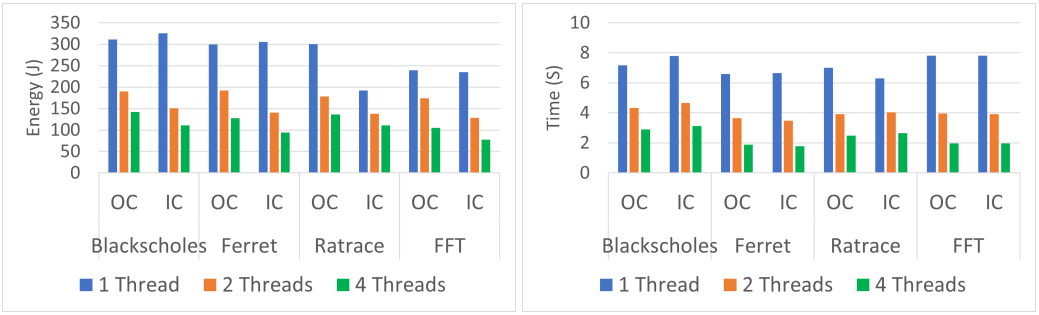


Figure 6. Containerization Effect on Energy and Time.

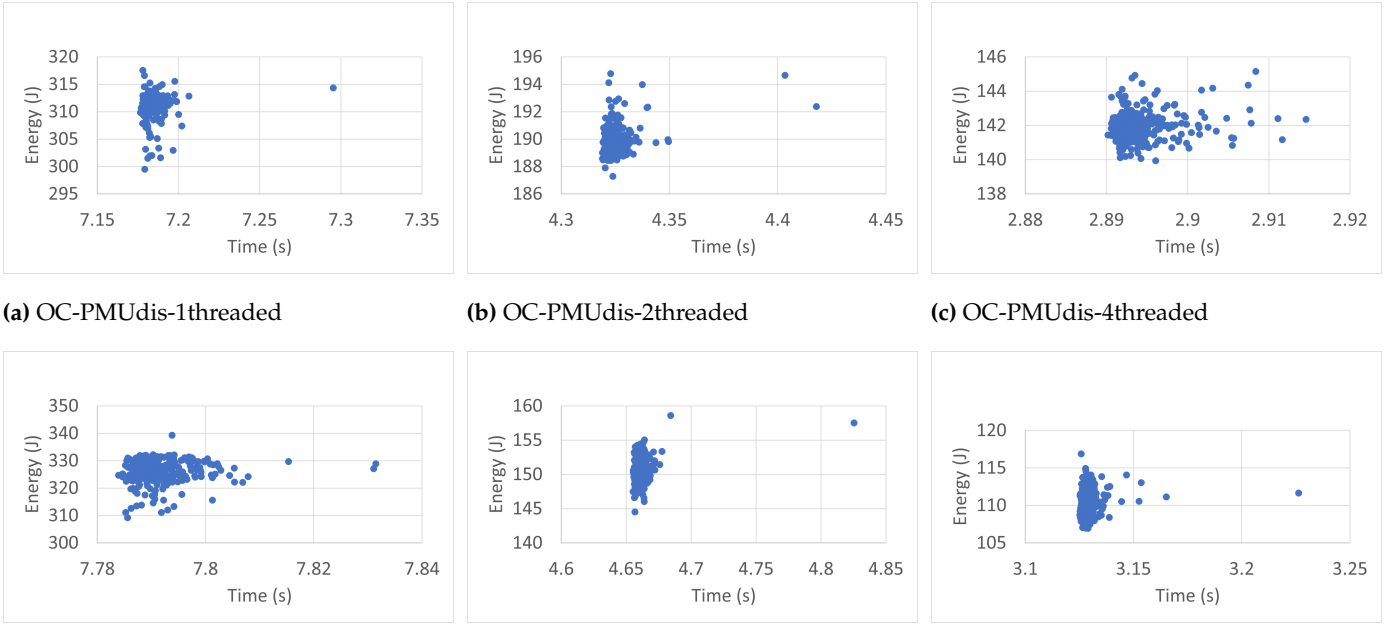
6.2. Measurements Accuracy

To estimate, to what extent, the accuracy of the measurements of the energy RAPL readings (300 readings), we studied the time-energy scatter plots of all the readings taken in the different conditions. The scatter plots, shown in Figures 7–14, were used to visualize the relationship between the energy and time variables and the resulting patterns. They display every RAPL reading pair (time, energy) as individual dot on the graph, with the time variable plotted on the horizontal axis and the energy variable plotted on the vertical axis.

Every plot caption represents a case where RAPL readings were collected. For instance, the caption: **Blackscoles-IC-PMUdis-2Threads**, means the scatter plot of the 300 readings taken for Blackscoles benchmark inside a docker container (IC) when the Power Management Unit is disabled (PMUdis) and the number of threads is 2.

To facilitate the comparison of the different cases, Figures 7–10 represent the scatter plots of the four benchmarks RAPL readings taken in the case of PMU is disabled. Every figure, in turn, is subdivided into six scatter plots arranged horizontally according to the

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 9 June 2023 doi:10.20944/preprints202306.0725.v1



(a) OC-PMUdis-1threaded **(b)** OC-PMUdis-2threaded **(c)** OC-PMUdis-4threaded

(d) IC-PMUdis-1threaded **(e)** IC-PMUdis-2threaded **(f)** IC-PMUdis-4threaded

Figure 7. Blackscoles Time-Energy Scatter Plots with Power Management Unit Disabled. Measurements are more condensed inside the container. Fewer threads seem to lead to more condensed readings.

number of threads and vertically according to whether the benchmark was run outside or inside the container. When we look at every figure horizontally, we can see the effect of the number of threads on the accuracy of readings. In most cases, the four thread plots are more scattered than one and two threads cases.

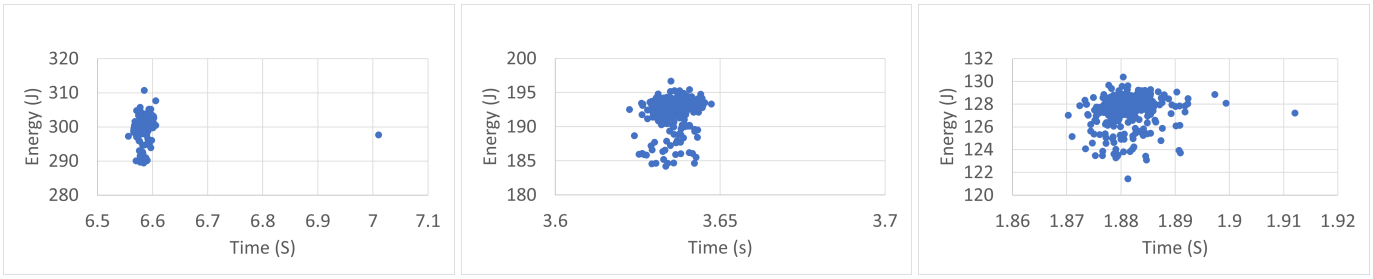
To study the effect of containerization on the scatter plots, one can look at every figure vertically. In some cases, there were similarities between the scatter plots of the same number of threads and run outside container (OC) or inside container (IC). However in most cases, the scatter plots of the IC cases are more condensed than that of the OC cases that represent more accurate RAPL readings inside containers.

Figures 11–14 represent the time-energy scatter plots of the four sets of RAPL readings taken in the case of one thread when the PMU was enabled, where every figure has two subfigures according to whether the readings were taken when the workload was run outside or inside the container.

In general, when the PMU is enabled, the scatter plots are more chaotic because of the dynamic control that the PMU applies to the CPU power, which may differ from one run to another, affecting, in turn, the accuracy of measurements. As the figures show, the scatter plots tend to be more than one cluster. This tendency makes taking the average of the measurements inaccurate. So, it is a good practice to turn off the PMU when we want to take the average of energy measurements to compare two or more codes from an energy perspective.

The characteristic of the workload may have a significant impact on the RAPL readings' accuracy. It is essential to consider the specific software characteristics and their effect on power consumption to obtain accurate and meaningful power consumption measurements using RAPL. One of these characteristics is the type of instructions executed within a workload, which impacts the repeated RAPL readings. For instance, complex instructions involving intensive calculations or data manipulations tend to consume more power when compared to simple instructions. Consequently, the repeated execution of workloads with computationally-complex instructions may result in higher average power consumption, which can reflect in the RAPL readings. Another example of the impact of workload

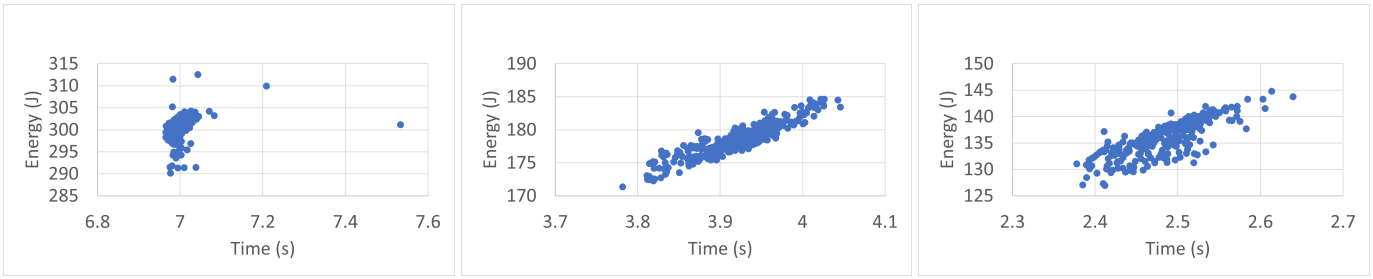
Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 9 June 2023 doi:10.20944/preprints202306.0725.v1



(a) OC-PMUdis-1threaded **(b)** OC-PMUdis-2threaded **(c)** OC-PMUdis-4threaded

(d) IC-PMUdis-1threaded **(e)** IC-PMUdis-2threaded **(f)** IC-PMUdis-4threaded

Figure 8. Ferret Time-Energy Scatter Plots with Power Management Unit Disabled. Taking readings with one thread inside and outside the container are more accurate than two and four threads.

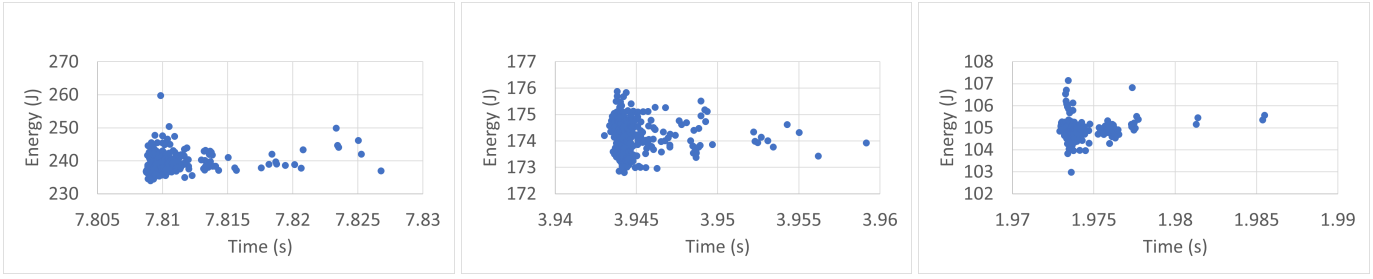


(a) OC-PMUdis-1threaded **(b)** OC-PMUdis-2threaded **(c)** OC-PMUdis-4threaded

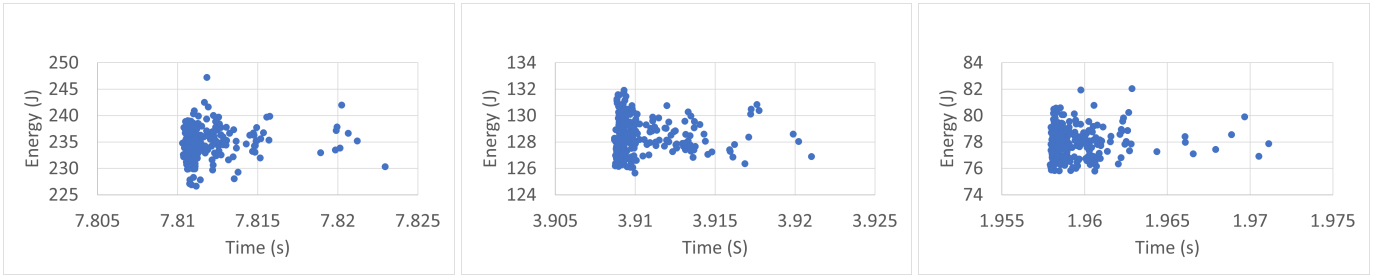
(d) IC-PMUdis-1threaded **(e)** IC-PMUdis-2threaded **(f)** IC-PMUdis-4threaded

Figure 9. Raytrace Time-Energy Scatter Plots with Power Management Unit Disabled. In this workload, readings for two and four threads are more accurate than those for one thread outside and inside the container.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 9 June 2023 doi:10.20944/preprints202306.0725.v1

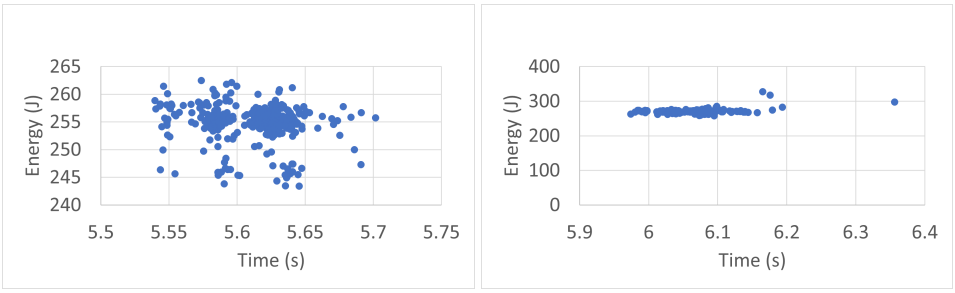


(a) OC-PMUdis-1threaded **(b)** OC-PMUdis-2threaded **(c)** OC-PMUdis-4threaded



(d) IC-PMUdis-1threaded **(e)** IC-PMUdis-2threaded **(f)** IC-PMUdis-4threaded

Figure 10. FFT Time-Energy Scatter Plots with Power Management Unit Disabled.



(a) OC-PMUdis-1threaded **(b)** IC-PMUdis-1threaded

Figure 11. Blackscholes Time-Energy Scatter Plot with Power Management Unit Enabled. The readings outside the container consist of two main clusters and three small clusters, which adversely affect the accuracy of the average value.

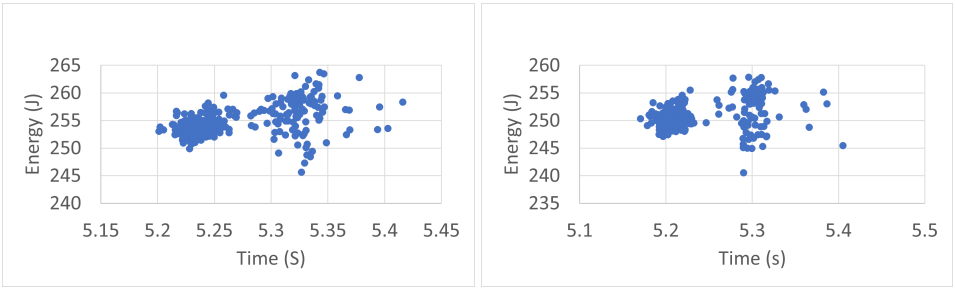
characteristics is the number of branch instructions in the workload. The outcome of a branch instruction determines which code path is executed next, potentially leading to different computational demands. Workloads with more branch instructions may exhibit more diverse execution patterns, resulting in varying power consumption levels and potentially impacting the accuracy of RAPL readings. Also, branch instructions can disrupt the instruction pipeline, impacting the efficiency of instruction fetching and execution. Mispredicted branches can lead to pipeline stalls and cache flushes, affecting the overall power consumption. Branch-heavy workloads may experience more frequent pipeline stalls and cache invalidations, potentially influencing the RAPL readings.

6.3. Experiences and Recommendations

Extensive experimentation provided valuable insights both into the environment and the measurement process. This section outlines the most important observations and recommendations based on lessons learned during the investigation.

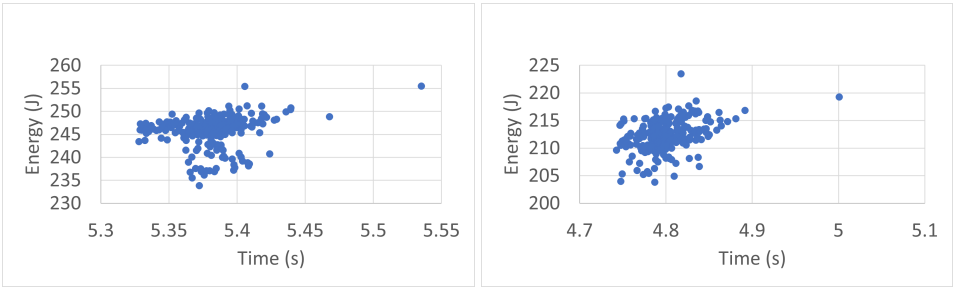
1. Readings seemed sensitive to the benchmark programs used for measurement. Researchers should be mindful of this point when devising ways to use RAPL to quantify

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 9 June 2023 doi:10.20944/preprints202306.0725.v1



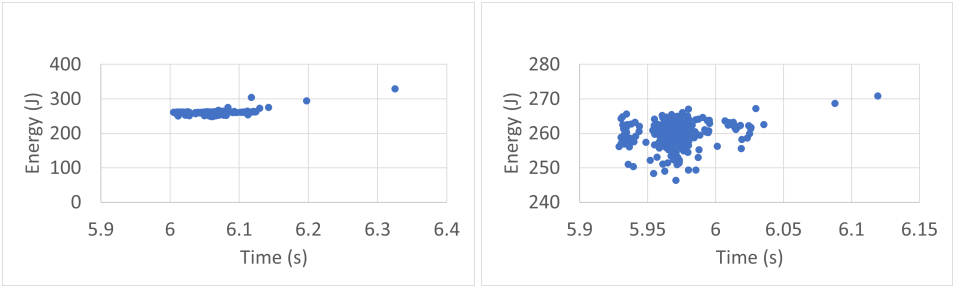
(a) OC-PMUdis-1threaded **(b)** IC-PMUdis-1threaded

Figure 12. Ferret Time-Energy Scatter Plot with Power Management Unit Enabled. These readings consist of two main cluster outside and inside container due to the control of the power management unit.



(a) OC-PMUdis-1threaded **(b)** IC-PMUdis-1threaded

Figure 13. Raytrace Time-Energy Scatter Plot with Power Management Unit Enabled.



(a) OC-PMUdis-1threaded **(b)** IC-PMUdis-1threaded

Figure 14. FFT Time-Energy Scatter Plot with Power Management Unit Enabled.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 9 June 2023 doi:10.20944/preprints202306.0725.v1

branch prediction behaviors. In particular, previous work had established that branch prediction schemes showed program sensitivity [41]. Therefore researchers should choose their test workload carefully to be able to discern behaviors due to branching from artifacts due to measurement.

2. Running software inside a container (e.g., the Docker container) may show a saving in RAPL energy readings despite the slight increase in the other event counts. The apparent savings may have resulted from including the libraries within the lighter-weight environment. Containers are lighter versions of full virtualization. They are also perhaps lighter than running directly under the OS. Containerization seemed to reduce environmental noise levels, which is conducive to the purposes of experimental studies. However, the effect of software containerization on energy consumption needs further investigation with more containers to confirm it.
3. Taking the average of many measurements of RAPL tends to be more accurate. However, increasing the number of readings beyond some point does not contribute to the accuracy. Instead, an enormous number may worsen matters due to the accumulative thermal effect.
4. The RAPL measurements are not very sensitive to the computer's ambient temperature. However, a good practice is to keep it consistent at a moderate level during RAPL readings collection.
5. Running software with multi-threading has a good impact on energy consumption and performance. However, measuring the average of RAPL readings with one thread seemed more accurate when comparing two or more pieces of code from an energy perspective.
6. Compiler optimization had a measurable impact on the RAPL readings. The experimental results showed that compiling software with the O0 optimization (no optimization beyond a conservative default set) always gives the highest energy consumption. With the other levels, results showed that more optimization is not always better for time and energy saving. However, to compare the energy consumption of two or more pieces of code without the unpredictable effects of the modifications typical of optimizing compilers, skipping them may be a good practice. It should help better understand the root causes of efficiency.
7. Internal power management had a positive impact on power saving. However, our experimental results showed that enabling the PMU harmed the accuracy of RAPL readings. This impact was probably due to the effects of the dynamic control of PMU on CPU power, which changed from one run to another. So, it is perhaps a good practice to disable the PMU from the BIOS before taking RAPL readings.

7. Conclusion

In computing systems, power/energy efficiency has become a crucial concern in the last few decades because of the various issues that require meeting a bunch of requirements, such as lowering environmental footprint, relieving the constraints on computing devices' scalability, the need for prolonging the life of computing energy storage devices, reducing the cost of the electricity bill and many other factors, including those related to security. Enhancing the quality jointly of hardware and software together can help achieve efficiency targets in those areas.

Some researchers were motivated by the need to reach exascale in reasonable power budgets. They investigated fundamental software building blocks, such as sorting and matrix multiplication, used commonly in HPC. In this study, the authors report on an investigation of a similar approach to DBP as a major hardware component in modern CPUs in an attempt to quantify its impact on power and energy efficiency. This paper focused on the long and tedious work to understand the issues involved and to develop the methodology to reliably estimate the power consumed by the DBP and its various aspects. It could also help as a blueprint for similar investigations targeting other hardware building-block components.

 doi:10.20944/preprints202306.0725.v1

RAPL is a helpful power measurement tool that can credibly replace external hardware power measurement devices that are difficult to manage and lack granularity. This paper introduced the methodology used in our investigation and the various factors that can affect the accuracy of RAPL measurements which add, inherently, noise to RAPL readings. To make RAPL more accurate and reliable, several measures that diminish or eliminate measurement noise sources were discussed in detail and supported by extensive experimentation. It included practical RAPL experimentation with four benchmarks that showed the effects of various hardware and software controls, such as processor power management, threading, containerization, and compiler optimization, on the RAPL readings. The techniques and experiences may also extend to other processor platforms that offer similar internal instrumentation and interface, such as those offered by AMD.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

HPC	High Performance Computing
DBP	Dynamic Branch Predictor
BTB	Branch Target Buffer
IBTB	Indirect Branch Target Buffer
RAPL	Running Average Power Limit
BM	Benchmark
1T	One Thread
2T	Two Threads
4T	Four Threads
OC	Outside Container
IC	Inside Container
PMUdis	Power Management Unit disabled
PMUen	Power Management Unit enabled
Mon	Mask-on
Moff	Mask-off

Appendix A.

Tables A1–A4 detail the data collected using the Linux `perf` tool. The numbers are the average of 300 run times for every workload. The data in Tables A1, A2, and A4 show thirteen CPU events for the four benchmarks selected from the PARSEC suite.

The data in table A1 was collected for two execution scenarios: a) on the host directly and b) inside a docker container. The cases are denoted in the table OC and IC, respectively. In each case, workloads were executed with one thread (1T), two threads (2T), and four threads (4T). In table A2, the date when the internal power management unit was enabled and disabled (cases marked in the table PMUen and PMUdis, respectively). In each of those cases, the workload was executed with one thread. Table A3 shows execution time and energy consumption collected when the upper side of the computer case was covered by a mask and with the mask off to measure the effect of ambient temperature on RAPL readings. The readings were taken when the internal power management unit was enabled (PMUen) and disabled (PMUdis). Table A4 details the events for two of the benchmarks measured when workloads were compiled with four levels of GCC compiler optimization, starting from O0 (minimal optimization) to O3 (aggressive optimization).

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 9 June 2023 doi:10.20944/preprints202306.0725.v1

Table A1. Events Measurement Data Inside and Outside Containers.

BM	Blackscholes						Ferret						Raytrace						FFT					
Events	OC			IC			OC			IC			OC			IC			OC			IC		
	1T	2T	4T	1T	2T	4T	1T	2T	4T	1T	2T	4T	1T	2T	4T	1T	2T	4T	1T	2T	4T	1T	2T	4T
EXT (S)	7.18	4.32	2.89	7.79	4.66	3.13	6.58	3.64	1.88	6.66	3.48	1.78	7	3.92	2.49	6.29	4.03	2.66	7.81	3.95	1.97	7.81	3.91	1.96
E(J)	310.99	189.76	141.93	325.9	150.50	110.55	299.65	192.23	127.38	305.12	140.35	94.29	300.59	178.57	136.17	192.34	137.82	110.64	239.41	174.17	104.89	234.8	128.35	77.85
P (W)	43.29	43.87	49.03	41.83	32.29	35.33	45.51	52.87	67.7	45.81	40.36	53.07	42.95	45.51	54.78	30.56	34.19	41.62	30.65	44.15	53.14	30.06	32.82	39.73
TC	36808	51981	34782	93646	56089	37659	79182	42253	21331	80123	42440	21477	84138	47191	29807	84264	48683	32094	93878	47398	23724	93901	47067	23576
#inst	32.74 B	32.53B	32.49B	34.26B	33.27B	33.90B	37.94B	37.3B	37.16B	39.33B	39.42B	39.58B	32.68B	32.95B	34.09B	32.77B	33.77B	35.94B	69.17B	69.42B	69.81B	71.55B	68.08B	69.20B
#Br	4.12B	4.08B	4.06B	4.08B	3.96B	3.97B	6.05B	5.94B	5.93B	6.23B	6.24B	6.33B	5.41B	5.44B	5.60B	5.42B	5.66B	6.21B	7.37B	7.41B	7.43B	7.62B	7.34B	7.34B
BrM	39.7M	31.23M	26.78M	43.04M	21.31M	20.59M	316.2M	326.46M	313.48M	317.78M	336.11M	325.08M	45.41M	38.76M	41.70M	45.62M	31.41M	40.34M	12.85M	21.56M	15.06M	13.53M	11.33M	10.32M
LLCL	11.66M	7.71M	5.92M	12.38M	4.87M	3.82M	122.93M	117.74M	118.3M	128.36M	111.59M	130.71M	12.33M	16.08M	27.11M	12.33M	13.49M	25.99M	4.08M	5.42M	2.30M	4.35M	2.20M	1.12M
LLCLM	0.255M	0.593M	0.591M	0.71M	0.422M	0.39M	12.09M	12.78M	16.74M	12.91M	14.37M	20.79M	87374	89285	48965	199247	51442	45920	206176	88630	39538	75001	41642	28695
LLCS	3.82M	2.36M	1.65M	4.06M	0.732M	3.82M	5.07M	1.956M	1.849M	10.42M	7.79M	8.24M	3.70M	10.13M	20.06M	3.67M	8.41M	18.65M	0.71M	1.94M	0.93M	0.86M	0.377M	0.18M
LLCSM	80037	64203	61432	77388	64924	390376	160834	2112561	2954441	1745861	382719	5419071	10312	6330	3138	10390	5055	5380	23025	6195	2046	12783	7663	3248
CS	3347	1494	1081	2899	2725	2017	9986	7670	6854	26131	24568	22715	3239	1642	45567	2642	2197	32551	2808	1794	826	4013	1976	1385
CPUM	161	64	49	133	133	106	202	198	709	386	106	213	151	70	43	116	109	95	138	73	36	190	102	71
PF	10979	11031	11144	11091	11055	10969	15838	16184	17248	26166	33844	65670	1469	1476	1536	1497	1361	1459	919	881	822	855	696	706

Table A2. Events Measurement Data with CPU Power Management ON/OFF.

BM	Blackscholes		Ferret		Raytrace		FFT	
Events	PMUen	PMUdis	PMUen	PMudis	PMUen	PMudis	PMUen	PMudis
EXT(S)	5.61	7.18	5.24	6.58	5.38	7	6.06	7.81
E(J)	254	310.99	250.90	299.65	245.8	300.59	260.27	239.41
P (W)	45.4	43.29	47.91	45.51	45.69	42.95	42.96	30.65
TC	68607	36808	63720	79182	65761	84138	73923	93878
#inst	32.59B	32.74 B	37.00B	37.94B	32.84B	32.68B	69.87B	69.17B
#Br	4.10B	4.12B	5.98B	6.05B	5.49B	5.41B	7.49B	7.37B
BrM	31.89M	39.7M	304M	316.2M	38.18M	45.41M	24.62M	12.85M
LLCL	10.20M	11.66M	126.25M	122.93M	11.39M	12.33M	9.3M	4.08M
LLCLM	0.75M	0.255M	13.96M	12.09M	0.21M	0.09M	210125	206176
LLCS	3.12	3.82M	4.38M	5.07M	2.84M	3.70M	2.93M	0.71M
LLCSM	0.16M	0.08M	132619	160834	12063	10312	21428	23025
CS	2335	3347	12591	9986	2522	3239	2912	2808
CPUM	102	161	199	202	106	151	125	138
PF	11093	10979	15883	15838	1487	1469	938	919

Table A3. Execution Time and Energy Data.

	Blackscholes				Ferret				Raytrace				FFT			
	PMUen		PMUdis		PMUen		PMUdis		PMUen		PMUdis		PMUen		PMUdis	
	Mon	Moff	Mon	Moff	Mon	Moff	Mon	Moff	Mon	Moff	Mon	Moff	Mon	Moff	Mon	Moff
EXT(S)	5.61	5.61	7.18	7.18	5.24	5.24	6.58	6.58	5.40	5.38	7.00	7.00	5.97	6.06	7.81	7.81
E(J)	254.60	254.72	310.35	310.99	254	250.90	303.04	299.65	251.08	245.80	301.35	300.59	259.39	260.27	235.86	239.41
P (W)	45.36	45.40	43.2	43.29	48.51	47.90	46.04	45.51	46.53	45.69	43.05	42.95	43.45	42.96	30.2	30.65

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 9 June 2023 doi:10.20944/preprints202306.0725.v1

Table A4. Effect of GCC Compiler Optimization Levels Data.

BM	Ferret				FFT			
Events	Optimization Level				Optimization Level			
	O0	O1	O2	O3	O0	O1	O2	O3
EXT(S)	17.45	7.74	6.58	7.49	29.08	7.13	7.81	7.47
E(J)	570.11	251.56	299.65	242.20	1172.2	286.10	239.41	299.20
P (W)	32.68	32.49	45.51	32.35	40.31	40.12	30.65	40.08
TC	193164.37	87428.93	79182	84514.79	349344.09	85629.40	93878	89699.84
#inst	109B	39.15B	37.94B	37.16B	153.29B	69.16B	69.17B	65.87B
#Br	7.98B	6.13B	6.05B	6.27B	8.14B	7.53B	7.37B	6.94B
BrM	383.04B	330.95M	316.2M	335.01M	95.14M	31.07M	12.85M	31.43M
LLCL	148.99M	118.14M	122.93M	118.27M	36.85M	8.77M	4.08M	9.41M
LLCLM	13.72M	10.51M	12.09M	10.81M	0.611M	0.17M	0.21M	0.17M
LLCS	5.30M	2.37M	5.07M	2.09M	13.82M	3.35M	0.71M	3.58M
LLCSM	140881	120906	160834	117213	59541	15610	23025	13184
CS	13895	7774	9986	7634	9442	2343	2808	2440
CPUM	443	219	202	216	458	111	138	120
PF	15984	15843	15838	15837	2036	892	919	902

References

1. Nain, S.; Chaudhary, P. Branch prediction techniques used in pipeline processors: a review. *International Journal of Pure and Applied Mathematics* **2018**, *119*, 2843–2851.
2. Hicks, M.A. Energy Efficient Branch Prediction. PhD thesis, University of Hertfordshire, 2007.
3. Kiriansky, V.; Waldspurger, C. Speculative buffer overflows: attacks and defenses. arXiv.1807.03757, 2018. https://doi.org/10.48550/arXiv.1807.03757.
4. Dong, X.; Shen, Z.; Criswell, J.; Cox, A.; Dwarkadas, S. Spectres, Virtual Ghosts, and Hardware Support. In Proceedings of the 7th International Workshop on Hardware and Architectural Support for Security and Privacy, Los Angeles California, 2 June 2018; HASP'18. https://doi.org/10.1145/3214292.3214297.
5. Kocher, P.; Horn, J.; Fogh, A.; Genkin, D.; Gruss, D.; Haas, W.; Hamburg, M.; Lipp, M.; Mangard, S.; Prescher, T.; et al. Spectre Attacks: Exploiting Speculative Execution. In Proceedings of the 2019 IEEE Symposium on Security and Privacy, San Francisco, CA, USA, 19–23 May 2019; SP, pp. 1–19.
6. da Silva, W.G.; Brisolara, L.; Corrêa, U.B.; Carro, L. Evaluation of the impact of code refactoring on embedded software efficiency. In Proceedings of the 1st Workshop de Sistemas Embarcados (WSE), 24–28 May 2010, SBRC'10, pp. 145–150.
7. Khan, K.N.; Ou, Z.; Hirki, M.; Nurminen, J.K.; Niemi, T. How much power does your server consume? estimating wall socket power using RAPL measurements. *Computer Science - Research and Development* **2016**, *31*, 207—214. https://doi.org/10.1007/s00450-016-0325-4.
8. Khan, K.N.; Hirki, M.; Niemi, T.; Nurminen, J.K.; Ou, Z. RAPL in action: experiences in using RAPL for power measurements. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)* **2018**, *3*, 1–26.
9. Mehta, H.; Owens, R.M.; Irwin, M.J.; Chen, R.; Ghosh, D. Techniques for low energy software. In Proceedings of the 1997 international symposium on Low power electronics and design, 1997, pp. 72–75.
10. Capra, E.; Francalanci, C.; Slaughter, S.A. Measuring application software energy efficiency. *IT Professional* **2012**, *14*, 54–61.
11. Sushko, S.; Chemeris, A. The dependence of microprocessor system energy consumption on software optimization. In Proceedings of the 2017 IEEE 37th International Conference on Electronics and Nanotechnology, IEEE, Kyiv, UKraine, 18-20 April 2017; ELNANO'17, pp. 451–454.
12. Al-Hashimi, M.; Saleh, M.; Abulnaja, O.; Aljabri, N. Evaluation of control loop statements power efficiency: An experimental study. In Proceedings of the 2014 9th International Conference on Informatics and Systems, Cairo, Egypt, 15-17 December 2014; INFOS'14, pp. PDC–45–PDC–48. https://doi.org/10.1109/INFOS.2014.7036676.
13. Abulnaja, O.A.; Ikram, M.J.; Al-Hashimi, M.A.; Saleh, M.E. Analyzing power and energy efficiency of bitonic mergesort based on performance evaluation. *IEEE Access* **2018**, *6*, 42757–42774. https://doi.org/10.1109/ACCESS.2018.2861571.
14. Aljabri, N.; Al-Hashimi, M.; Saleh, M.; Abulnaja, O. Investigating power efficiency of mergesort. *Journal of Supercomputing* **2019**, *75*, 6277–6302. https://doi.org/10.1007/s11227-019-02850-5.

15. Al-Hashimi, M.; Saleh, M.; Abulnaja, O.; Aljabri, N. On the power characteristics of mergesort: An empirical study. In Proceedings of the 2017 Int'l Conf. on Advanced Control Circuits and Systems & 2017 Int'l Conf. on New Paradigms in Electronics & Information Technology, IEEE, Alexandria, Egypt, 5-8 November 2017; ACCS'17/PEIT'17, pp. 172–178.
16. Oo, N.Z.; Chaikan, P. The Effect of Loop Unrolling in Energy Efficient Strassen's Algorithm on Shared Memory Architecture. In Proceedings of the 2021 36th International Technical Conference on Circuits/Systems, Computers and Communications, Jeju, South Korea, 27-30 June 2021; ITC-CSCC, pp. 1–4.
17. Jammal, F.; Aljabri, N.; Al-Hashimi, M.; Saleh, M.; Abulnaja, O. A preliminary empirical study of the power efficiency of matrix multiplication. *Electronics* **2023**, *12*.
18. Lastovetsky, A.; Manumachu, R.R. Energy-efficient parallel computing: challenges to scaling. *Information* **2023**, *14*. https://doi.org/10.3390/info14040248.
19. Emma, P.G.; Davidson, E.S. Characterization of branch and data dependencies in programs for evaluating pipeline performance. *IEEE Transactions on Computers* **1987**, *C-36*, 859–875. https://doi.org/10.1109/TC.1987.1676981.
20. Mittal, S. A survey of techniques for dynamic branch prediction. *Concurrency and Computation: Practice and Experience* **2019**, *31*, e4666.
21. Lin, C.K.; Tarsa, S.J. Branch Prediction Is Not A Solved Problem: Measurements, Opportunities, and Future Directions. In Proceedings of the 2019 IEEE International Symposium on Workload Characterization, Orlando, FL, USA, 3-5 Nov 2019; IISWC'19, pp. 228–238. https://doi.org/10.1109/iiswc47752.2019.9042108.
22. Uzelac, V.; Milenkovic, A. Experiment flows and microbenchmarks for reverse engineering of branch predictor structures. In Proceedings of the 2009 IEEE International Symposium on Performance Analysis of Systems and Software, Boston, MA, USA, 26-28 April 2009; pp. 207–217. https://doi.org/10.1109/ISPASS.2009.4919652.
23. Desrochers, S.; Paradis, C.; Weaver, V.M. A validation of DRAM RAPL power measurements. In Proceedings of the Second International Symposium on Memory Systems, Alexandria, VA, USA, October 3 - 6 2016; MEMSYS '16, pp. 455–470.
24. David, H.; Gorbatov, E.; Hanebutte, U.R.; Khanna, R.; Le, C. RAPL: Memory power estimation and capping. In Proceedings of the 16th ACM/IEEE International Symposium on Low Power Electronics and Design, Austin, Texas, USA, 18-20 Aug 2010; ISLPED '10, pp. 189–194.
25. Hähnel, M.; Döbel, B.; Völp, M.; Härtig, H. Measuring energy consumption for short code paths using RAPL. *ACM SIGMETRICS Performance Evaluation Review* **2012**, *40*, 13–17.
26. Giardino, M.; Ferri, B. Correlating hardware performance events to CPU and DRAM power consumption. In Proceedings of the 2016 IEEE International Conference on Networking, Architecture and Storage (NAS), Long Beach, CA, USA, 08-10 August 2016; pp. 1–2.
27. Zhang, H.; Hoffman, H. A quantitative evaluation of the RAPL power control system. In Proceedings of the 10th International Workshop on Feedback Computing, Seattle, WA, USA, April 13th 2015; Feedback Computing 15.
28. Khan, K.N.; Ou, Z.; Hirki, M.; Nurminen, J.K.; Niemi, T. How much power does your server consume? Estimating wall socket power using RAPL measurements. *Computer Science - Research and Development* **2016**, *31*, 207–214.
29. Hsu, R.C.; Liu, C.T.; Wang, H.L. A reinforcement learning-based ToD provisioning dynamic power management for sustainable operation of energy harvesting wireless sensor node. *IEEE Transactions on Emerging Topics in Computing* **2014**, *2*, 181–191.
30. Shivam, A. *A Multiple Compiler Approach for Improved Performance and Efficiency*; University of California, Irvine, 2021.
31. Chen, B.; Nedelchev, I. Power compiler: a gate-level power optimization and synthesis system. In Proceedings of the International Conference on Computer Design VLSI in Computers and Processors, IEEE, Austin, TX, USA, 12-15 October 1997; pp. 74–79.
32. TURBOSTAT(8) - System Manager's Manual. https://www.linux.org/docs/man8/turbostat.html.
33. Ubuntu Manpage: powertop - a power consumption and power management diagnosis tool. https://manpages.ubuntu.com/manpages/bionic/man8/powertop.8.html.
34. `perf`: Linux profiling with performance counters. https://perf.wiki.kernel.org.
35. Zakharenkov, R. DevOps in E-commerce software development: Demand for Containerization. Technical report, Oulu University of Applied Sciences, 2019.
36. Xu, C.; Zhao, Z.; Wang, H.; Shea, R.; Liu, J. Energy efficiency of cloud virtual machines: from traffic pattern and CPU affinity perspectives. *IEEE Systems Journal* **2015**, *11*, 835–845.

 doi:10.20944/preprints202306.0725.v1

37. Linux Kernel in a Nutshell. https://www.linuxtopia.org/online_books/linux_kernel/kernel_configuration/re46.html. Scheduler options - Chapter 10.
38. Boyd-Wickizer, S.; Chen, H.; Chen, R.; Mao, Y.; Kaashoek, F.; Morris, R.; Pesterev, A.; Stein, L.; Wu, M.; Dai, Y.; et al. Corey: An Operating System for Many Cores. In Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation, San Diego, California, 8 December 2008; OSDI'08, pp. 43—-57.
39. Bienia, C.; Kumar, S.; Singh, J.P.; Li, K. The PARSEC benchmark suite: Characterization and architectural implications. In Proceedings of the 17th international conference on Parallel architectures and compilation techniques, Toronto Ontario Canada, 25–29 October 2008; PACT'08, pp. 72–81. https://doi.org/10.1145/1454115.1454128.
40. Colmant, M.; Rouvoy, R.; Kurpicz, M.; Sobe, A.; Felber, P.; Seinturier, L. The next 700 CPU power models. *Journal of Systems and Software* **2018**, *144*, 382–396.
41. Smith, J.E. A Study of Branch Prediction Strategies. In Proceedings of the 8th Annual Symposium on Computer Architecture, Minneapolis, Minnesota, USA, May 12–14 1981; ISCA'81, pp. 135—-148. https://doi.org/10.5555/800052.801871.