The Price of Meltdown and Spectre: Energy Overhead of Mitigations at Operating System Level

Benedict Herzog
Friedrich-Alexander-Universität Erlangen-Nürnberg

Stefan Reif
Friedrich-Alexander-Universität Erlangen-Nürnberg

Julian Preis
Friedrich-Alexander-Universität Erlangen-Nürnberg

Wolfgang Schröder-Preikschat
Friedrich-Alexander-Universität Erlangen-Nürnberg

Timo Hönig
Ruhr-Universität Bochum

ABSTRACT

The Meltdown and Spectre hardware vulnerabilities shocked hardware manufacturers and system users upon discovery. Numerous attack vectors and mitigations have been developed and deployed. However, due to the deep entanglement in core CPU components they will be an important topic for years. Although the performance overhead of software mitigations has been examined closely, the *energy overhead* has experienced little attention—even though the energy demand is a critical cost factor in data centres.

This work contributes a fine-grained energy-overhead analysis of Meltdown/Spectre software mitigations, which reveals application-specific energy overheads of up to 72 %. We further compare energy overheads to execution time overheads. Additionally, we examine subsystem-specific effects (i.e., CPU, memory, I/O, network/inter-process communication) and develop a model that predicts energy overheads for applications.

ACM Reference Format:
Benedict Herzog, Stefan Reif, Julian Preis, Wolfgang Schröder-Preikschat, and Timo Hönig. 2021. The Price of Meltdown and Spectre: Energy Overhead of Mitigations at Operating System Level. In *14th European Workshop on Systems Security (EuroSec'21), April 26, 2021, Online, United Kingdom.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3447852.3458721

1 INTRODUCTION

The discovery and publication of the Meltdown [13] and Spectre [11] hardware vulnerabilities in 2018 shocked both hardware manufacturers as well as computer system users. Since then, numerous new attack vectors have been found, and mitigations at hardware, firmware, and software level have been developed and deployed at a large scale [7, 11, 13]. As these vulnerabilities lie within the heart of modern CPU's security features, they will nevertheless keep haunting us. Similar attacks and mitigations [3, 4] are likely an important topic for the years to come.

Since the inception of (software-level) mitigations, their performance impact aroused great interest [7]. Due to the deep entanglement within core CPU components (e.g., branch prediction units, speculative execution), the performance hit has often been severe. Over time, noticeable research has been done to analyse and potentially reduce these overheads [1, 17, 18]. However, only little research has been done to analyse the *energy demand* overhead of these mitigations. Especially for cloud service providers, their systems' energy demand is one of the most critical cost factors—on the one hand for the operation of their systems themselves and, on the other hand, for the cooling efforts to lead away thermal dissipation [6]. In addition, the energy demand of computing systems has become a crucial non-functional property for mobile devices (e.g., laptop computers) and in times of *Green IT.* Although the energy demand and execution time of software correlate often, previous research has shown that this is not always the case [2].

The goal of this work is to *put a price tag* on the Meltdown/Spectre mitigations in terms of their energy overhead. Ideally, this allows system operators and users to reason about the associated overheads and weigh up the reduced attack vector against increased operational and environmental costs. For example, in scenarios with only trustworthy software or relaxed security requirements, no or at least not all software-level mitigations are required. This principle to not burden users (and the environment nowadays) with unneeded features was already described by Parnas in 1979[1] [16]. This work provides the necessary information basis for this assessment process. Furthermore, professional operators often have in-depth knowledge about their application, for example, the amount of disk I/O or communication an application usually performs. As the mitigations affect individual subsystems to different extents, this knowledge can help to estimate the impact of selectively activating or deactivating Meltdown/Spectre mitigations.

One difficulty, however, is the determination of energy overheads. Measuring energy (or power) is a tedious task requiring measurement interfaces or measurement devices and human resources to conduct measurements. Furthermore, the energy demand depends on various factors such as background noise, temperature, and specific hardware. To relieve providers and users from actual measurements, energy models have been proven suitable to predict software's energy demand with low effort [9, 15, 20] and, importantly, ahead of execution. Hence, a model to predict the energy

EuroSec'21, April 26, 2021, Online, United Kingdom
© 2021 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-8337-0/21/04…$15.00
https://doi.org/10.1145/3447852.3458721

[1] "Some users may require only a subset of the services or features that other users need. These *less demanding* users may demand that they are not be forced to pay for the resources consumed by the unneeded features." [16]

overhead for an application can help service providers to reduce their costs and allows an easier trade-off between security level and operational costs for users.

To provide operators and users with the required knowledge, this papers examines the following four research questions:

- **Q1** How much energy overhead for applications is introduced by Meltdown/Spectre mitigations?
- **Q2** Is the energy overhead of Meltdown/Spectre mitigations related to the extensive use of specific subsystems (i.e., CPU, memory, block I/O, operating system interactivity, and communication)?
- **Q3** Is the energy overhead correlated with the execution time overhead?
- **Q4** Is the energy overhead predictable for a given application?

In order to answer these questions, this paper is structured as follows. Section 2 gives a short summary of the Meltdown and Spectre vulnerabilities. In Section 3 our measurement methodology and the implementation of our prediction model is described. The analysis of the mitigations' energy demand footprint is presented in Section 4, thereby questions **Q1**, **Q2**, and **Q3** are examined. Section 5 evaluates our prediction model and thereby considers **Q4**. Finally, Section 6 presents related work and Section 7 concludes this paper.

2 MELTDOWN AND SPECTRE

The Meltdown and Spectre vulnerabilities introduced an entirely new class of attacks on modern CPU architectures. Both exploit side channels at hardware level to reconstruct memory contents without the required access permissions. As these vulnerabilities relate to hardware, new processor revisions can provide full mitigation. For existing vulnerable CPUs, software mitigations hinder or avoid specific variants, but do so only partly or with high overheads [1, 17, 18]. This section shortly describes the vulnerabilities and deployed mitigations.

Meltdown. The Meltdown [13] attack utilises a race condition between the execution of a CPU instruction and the corresponding privilege check. If an instruction has already been executed despite not having the required privileges, the CPU voids all functional instruction effects. Although functional effects are thereby reversed, non-functional effects, for example, caching effects of accessed memory, still may be observable. These caching effects can be used to reconstruct the original memory contents and thus bypass the privilege check. This attack is applicable for all mapped virtual memory irrespective of the applied memory access protections. Therefore, it allows a malicious process to read arbitrary kernel memory and data structures. Usually, modern kernels (e.g., Linux) map all physical memory for easy access and thus allow attackers to access all installed memory.

The Linux kernel mitigates Meltdown attacks by use of *Kernel Page Table Isolation* (KPTI). Thereby, the kernel manages two page table versions per process. One only used in kernel mode, where all memory is mapped, and one used while in user mode, where only the user-space memory and a minimal kernel memory region for system calls and interrupts is mapped. This introduces additional overhead for privilege level switches between user and kernel mode for changing the page table version and potential (selective) flushes of the *Translation Lookaside Buffer* (TLB).

Spectre. The Spectre [11] attacks use a similar technique as Meltdown attacks, but do not rely on a race condition between privilege check and memory access, but utilise the speculative execution and branch prediction of modern CPUs. If a branch is executed speculatively and the CPU detects that the taken branch was mispredicted, the CPU voids all functional effects. However, comparable to Meltdown, non-functional effects still can be observable and used to reconstruct data used during the speculative execution. Because the branch prediction unit of a CPU is a per-core entity, a malicious process can pre-train the branch prediction unit to dependably mispredict the branch on a target process. Afterwards, the malicious process can use the timing behaviour of memory accessed by the mispredicted branch to reconstruct the target process's memory contents. Hence, malicious processes can reconstruct the memory of arbitrary processes (including kernel threads) running on the same CPU core.

As Spectre describes a new class of attacks rather than a single attack or vulnerability, there is no single mitigation against Spectre. Instead, there is a collection of mitigations against specific variants of Spectre attacks deployed. Examples relevant for this paper are `swapgs` barriers, retpolines, *Indirect Branch Restricted Speculation* (IBRS), *Return Stack Buffer* (RSB) refilling. All of these mitigations introduce—to different extents—overhead, whereby the specific overhead depends on the application.

3 IMPLEMENTATION

In order to quantify the mitigations' overheads, we conduct execution time and energy measurements for different sets of benchmarks and mitigation activation states. Furthermore, we develop a prediction model, based on linear regression, for energy overhead estimations.

3.1 Measurement Methodology

We conducted our time and energy measurements on an off-the-shelf desktop computer. It utilises an Intel Core i5-8400 CPU running at 2.8 GHz (Turbo boost 4.0 GHz), which is vulnerable to Meltdown and Spectre attacks. The computer is equipped with 8 GB of RAM, a 2 TB hard disk drive, and a standard 1 Gbit ethernet interface. The system software is the Ubuntu 18.04 LTS distribution built upon the Linux kernel version 4.15.

For energy measurements, we utilise Intel's *Running Average Power Limit* (RAPL) mechanism. It allows continuous energy measurements of CPU components with high accuracy at a sampling rate of 1 ms (as documented by Intel) or even faster (as documented by Lipp et al. [12]). We measure the energy demand at the *package* level, which includes the whole CPU package (including all cores, the uncore, and internal GPU). During measurements, the Linux *powersave* governor is active.

As RAPL measures the CPU's energy demand, it allows an accurate analysis of the mitigations' effects on the CPU. Furthermore, RAPL is available on a wide variety of Intel CPUs and thus allows an easy repetition and comparison of our analysis on different Intel hardware platforms. Usually, higher CPU activity (and thus energy demand) also leads to higher system energy demand. However, the correlation between CPU's and system's energy demand is not always linear. Hence, in this work we focus mainly on the mitigations' effects on the CPU's energy demand.

Both energy and time measurements are conducted using the *perf* tool. For each measurement, we conduct ten iterations to minimise the impact of outliers and noise. As the energy behaviour for benchmarks including block I/O and network communication are more unstable, we ran an additional 40 iterations for benchmarks in the *I/O* and *Communication* group (cf. Figure 1). To reduce caching effects, all benchmarks are executed intermingled rather than in a loop. Additionally, we clean kernel caches before the execution of benchmarks in the I/O and Communication group[2].

We analyse the energy overheads of Meltdown/Spectre mitigations employing 28 benchmarks in total. Our benchmark set consists of four benchmarks from the *sysbench* benchmark suite[3] and 24 benchmarks from the *stress-ng* benchmark suite[4]. To evaluate the quality of the overhead prediction model we use a separate set of ten benchmarks from the *Phoronix* test suite [5]. The benchmarks' execution times lie between 20 s and 300 s for the training benchmarks with one exception[6]. This constitutes a good trade-off between reproducible results, low measurement overhead, and moderate measurement efforts. The execution times for the evaluation benchmarks lie between 2 min-15 min.

3.2 Linux Mitigations

As described in Section 2, Linux implements several mitigations against Meltdown and Spectre. The mitigations can be enabled and disabled at boot time via the kernel command line interface. In this paper, we consider the `nopti`, `nospectre_v1`, and `nospectre_v2` command line options. The `nopti` option disables KPTI used to mitigate Meltdown attacks. The `nospectre_v1` option disables `swapgs` and usercopy barriers and disables pointer sanitisation. The `nospectre_v2` option disables retpolines, IBRS, and RSB filling.

3.3 Prediction Model

To predict the energy overhead for Meltdown/Spectre mitigations, we have developed a linear model, using supervised learning. The model is trained on the data of the 28 benchmarks (cf. Section 3.1) and evaluated on the ten benchmarks of the Phoronix test suite.

Features and Labels. We selected four performance counters (PMCs) as inputs for the prediction model. This small number of PMCs ensures that all their values can be obtained in a single measurement iteration with the perf tool.

(1) instructions per cycle (IPC)
(2) branches per cycle (BPC)
(3) system calls per million cycles (SCPMC)
(4) process context switches per million cycles (CSPMC)

The goal of our selection is to model the degree of system interactivity of an application and to cover relevant areas of the Meltdown/Spectre attacks (e.g., KPTI and the branch prediction unit). The output of the model is the energy overhead. Previous work showed that PMCs are subject to variations between several measurements runs due to non-determinism of, for example, operating system activity and execution environment (e.g., memory pressure) [5]. However, this non-deterministic effects also affect the energy demand and thus it is important to include these in our model. We account for this by conducting several measurement runs and using the mean for the collected PMCs.

Model Training. After the collection of all energy demand and PMC data, we have trained a linear model to predict the energy overhead from the collected PMCs. For modelling, we utilised the linear regression implementation of the *sklearn* framework[7].

4 ENERGY OVERHEAD ANALYSIS

This section analyses the measurement data for our benchmark set consisting of 28 benchmarks in total. First, we examine the energy overheads for different combinations of mitigations (cf. Section 4.1, **Q1**). Second, we analyse the impact of different subsystems (cf. Section 4.2, **Q2**). Third, we investigate whether the energy and execution time overheads are correlated (cf. Section 4.3, **Q3**).

4.1 Energy Footprint of Mitigations

Figure 1 gives an overview of the mitigations' respective overhead compared to the execution with all mitigations disabled. We analyse the energy overhead for four cases: all mitigations enabled (`all-enabled`) and all but one mitigation enabled (`no kpti`, `no spectre-v1`, and `no spectre-v2`). Hence, we can quantify the total overhead, and also identify the mitigation(s) introducing the overhead. We calculate the respective overheads using the geometric mean energy demand normalised to the geometric mean energy demand with all mitigations disabled. The error bars indicate the positive and negative 95 % confidence intervals of this overhead to assess the variability. In addition, we visualise the positive and negative 95 % confidence intervals of the measurements with `all disabled` mitigations as grey background boxes.

Figure 1 shows mixed results for the energy overheads. On the one hand, many benchmarks have only minor overheads. In total, 11 out of 28 benchmarks have an overhead below 5 %. On the other hand, we observe significant overheads of up to 72 %. In total, eight benchmarks that have an overhead greater than 25 %. This leaves nine benchmarks with a medium overhead between 5 % and 25 %.

One interesting finding is that KPTI often has the greatest influence on the overhead. For example, the `icache` benchmark has only a small overhead (3 %) if KPTI is disabled. If KPTI is enabled, the overhead rises to 49 %. Similar behaviour can be seen for other benchmarks (e.g., `aio`, `threads`, `context`) as well.

In some cases, however, not only KPTI, but also Spectre v2 mitigations introduce noticeable overhead. For example, for the `switch` and `pipe` benchmarks disabling the Spectre v2 mitigations significantly reduces the overhead. Disabling the Spectre v1 mitigations, however, only slightly reduces the overhead in most cases.

In summary, our analysis shows that it depends on the application, whether the Meltdown/Spectre mitigations have a significant impact on the energy demand. We observed both, almost no overhead at all as well as substantial overheads of up to 72 %. For

[2]`echo 1 > /proc/sys/vm/drop_caches`
[3]https://github.com/akopytov/sysbench (v.1.0.11)
[4]https://kernel.ubuntu.com/~cking/stress-ng/ (v.0.09.25)
[5]https://www.phoronix-test-suite.com/ (v.9.6.1)
[6]The maximum configurable execution time for the `branch` benchmark is 1 s-2 s
[7]https://scikit-learn.org

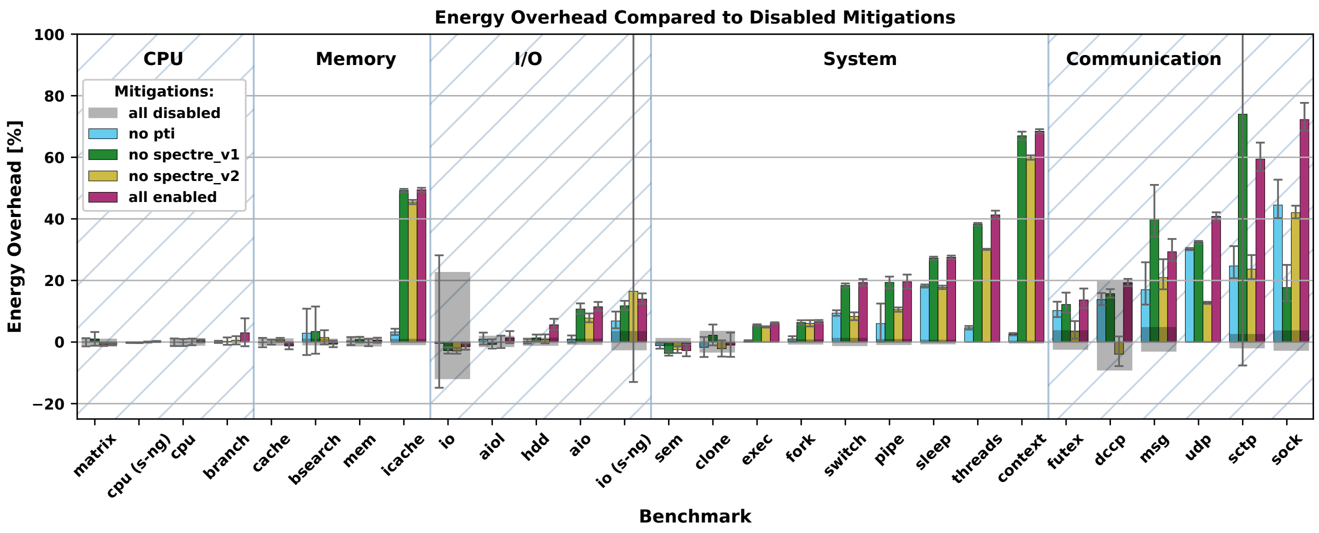

Figure 1: Energy overhead for Meltdown/Spectre mitigations. The baseline is measured with all mitigations disabled. The grey boxes indicate the 95 % confidence intervals for the baseline to assess the benchmarks' variance. The bars represent the energy overhead normalised to the baseline in percent. The error bars indicate the positive and negative 95 % confidence intervals.

applications with an overhead, disabling KPTI often reduces this overhead greatly or even completely. In some cases, the overhead is introduced by both KPTI and mitigations against Spectre v2. Disabling Spectre v1 mitigations yields only minor overhead savings in most cases.

4.2 Subsystem-specific Energy Footprint

This section further analyses the subsystem-specific overheads introduced by Meltdown/Spectre mitigations. We categorised the benchmark set in groups depending on how heavily a specific subsystem is used. Our categories are CPU, memory (i.e., intensity of memory accesses), system (i.e., context switches), block I/O, and communication (i.e., network and inter-process communication). However, this is not a sharp categorisation as one benchmark may fit into several categories but nonetheless allows identifying subsystem-specific effects.

Our first finding is that for our CPU-, memory-, and I/O-heavy benchmarks the Meltdown/Spectre mitigations have no to a relatively small overhead with only one exception—the `icache` benchmark shows over 40 % overhead if KPTI is enabled.

In general, we attribute the overhead for KPTI to the fact that switching from user to kernel mode is significantly more expensive if KPTI is enabled [18]. The CPU and memory benchmarks mainly stress the specific subsystem and do not interact much with the operating system. Thus, these groups are less affected by the overhead of KPTI. As an exception, the `icache` benchmark continuously flushes the instruction cache and therefore issues many system calls. The high system call rate is the reason why it is highly affected by KPTI. The I/O benchmarks also trigger system calls for I/O actions, however, only need a small number of actions to trigger I/O activity and the I/O costs dominate the system call costs. Hence, the mitigations introduce only a relatively small overhead.

As expected, with more operating system interactivity in the *system* and *communication* category, more benchmarks have higher overheads. Often the main overheads result from KPTI (e.g., `fork`, `exec`, `threads`, `context`). However, there are some benchmarks where the mitigations against Spectre v2 also introduce overheads, for example, `switch`, `sleep`, and `futex`. In these cases, the overhead results from the combination of mitigations. We plan further to investigate the negative overheads for `dccp` and `io`. However, both benchmarks show a great variance even if all mitigations are disabled due to the network benchmarks' dynamic behaviour and the effect may be measurement noise.

In summary, our analysis shows that mitigations can introduce a significant overhead for applications that interact with the operating system frequently, especially KPTI and, to a smaller extent, Spectre v2. However, the Meltdown/Spectre mitigations often have no significant impact on the energy demand for applications with little system interactivity, like CPU- or memory-heavy applications. Similarly, for applications performing many block I/O operations, the I/O costs usually dominate the costs for mitigations.

4.3 Time and Energy Correlation

In many cases, the energy demand, especially the CPU's energy demand, and execution time of an application are correlated [8]. However, previous research has shown that this correlation not always applies [10]. Hence, this section analyses the correlation between energy and execution time overhead if all Meltdown/Spectre mitigations are enabled.

Figure 2 visualises the energy and time overheads for our benchmark set with all mitigations enabled using the geometric mean of all measurements (cf. Section 4.1). The figure shows that the execution time overhead for most benchmarks is highly correlated with the energy overhead, especially for benchmarks with small energy

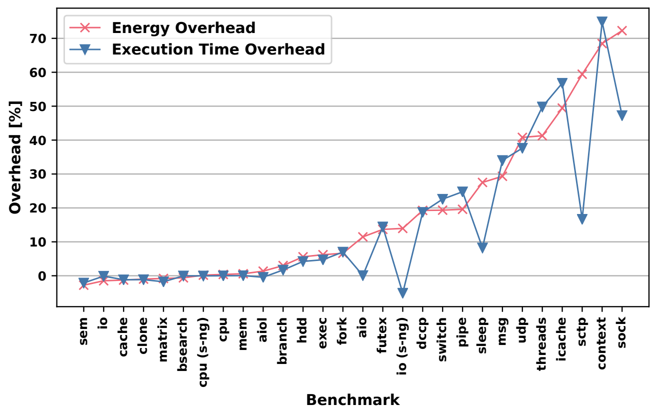

Figure 2: Energy and execution time overhead if all Meltdown/Spectre mitigations are active.

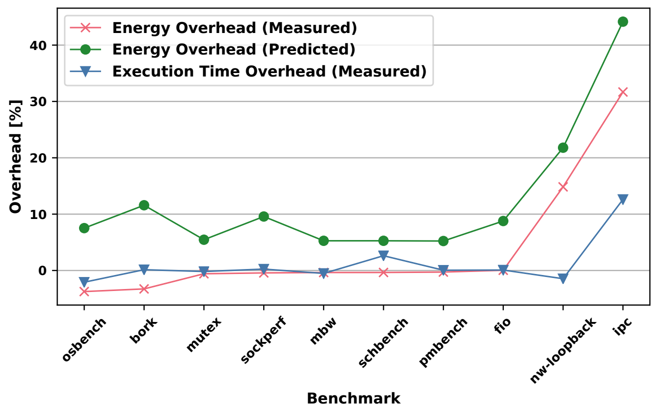

Figure 3: Energy overheads for the Phoronix benchmarks as measured (red) and as predicted by the linear model (green). Additionally, the measured execution time overhead is shown (blue).

overhead. In total, we observe a Spearman correlation coefficient of 0.88 (i.e., a strong positive correlation). However, there are some noticeable exceptions—the `aio`, `io (s-ng)`, `sleep`, `sctp`, and `sock` benchmarks. For these benchmarks, the execution time overhead is not related to the energy overhead. As the benchmarks' execution times (20 s-300 s) are significantly higher than RAPL's sampling rate (less than 1 ms) and the results are reproducible, these findings are no measurement artefacts. Additionally, these exceptional benchmarks stem from different subsystem categories, that is, from the memory, I/O, system, and communication category. Hence, the reason for these exceptions lies not within a specific subsystem, but within the benchmarks. However, using the PMC data we could not identify the underlying reason for these increased energy overheads. Hence, a further analysis of these benchmarks, for example at instruction level, is necessary.

In summary, the execution time overhead is, in general, correlated with the energy overhead and can give helpful hints on the energy overhead introduced by Meltdown/Spectre mitigations. However, there are noticeable exceptions (five out of 28) where the energy overhead does not match the execution time overhead. Additionally, as described in Section 3.1, our energy measurements only cover the CPU package energy demand. Therefore, energy overhead for peripheral devices (e.g., hard disk drives or network interfaces) can occur, especially for the I/O and communication categories. We plan to analyse energy overheads of peripheral devices in future work.

5 ENERGY OVERHEAD PREDICTION

The evaluation of our prediction model of energy overheads consists of two parts: (1) an analysis of whether our model input features are correlated with the energy overhead and (2) a comparison between predicted and measured overheads for a set of applications, which were not used for model training, to evaluate the model prediction quality and thus answer **Q4**.

Table 1 shows the Spearman correlation coefficients between the energy overhead and the four performance counters (PMCs) for the sysbench and stress-ng benchmark suite. Additionally, the coefficients between execution time overhead and the PMCs are depicted. For this correlation analysis, we compare the `all enabled` setting to `all disabled`.

	Performance Counters			
Overhead	**IPC**	**BPC**	**SCPMC**	**CSPMC**
Energy	-0.06	-0.02	0.64	0.41
Execution Time	-0.02	-0.03	0.64	0.33

Table 1: Spearman correlation coefficient between performance counters and energy/execution time overhead with all mitigations active for the sysbench and stress-ng benchmark suite.

On the one side, the IPC and BPC performance counters are uncorrelated with both the energy and the execution time overhead. Thus, in future models, the IPC and BPC could be replaced by other features with more influence. On the other side, system calls and context switches correlate with both the energy and execution time overhead. Thereby, the correlation for SCPMC is stronger than for CSPMC. The correlation coefficients for energy and execution time overhead are similar, which confirms our finding in Section 4.3 that the overheads correlate and have similar magnitudes.

Figure 3 illustrates the measured and predicted energy overheads for the Phoronix benchmarks if all mitigations are enabled. Additionally, the measured execution time overheads are shown. The measured energy and execution time overheads confirm the previous section's findings that the overheads are highly application-specific (between no up to ~30 % energy overhead). Especially interesting is the `nw-loopback` benchmark, where we measure a significant energy overhead (~15 %) but no execution time overhead. In general, it can be noted that the Phoronix benchmarks have fewer energy overheads than the more synthetic benchmarks used in Section 4. As the benchmarks in Section 4 are selected to trigger extreme cases and identify sources of overheads and the Phoronix benchmarks in this section are more real-world benchmarks, this is in line with our expectations.

Our linear model overestimates the energy overhead by around 5 %. We attribute this behaviour to the training data (i.e., the benchmarks in Section 4), which are more synthetic and trigger extreme cases (e.g., purely making system calls), and the simplicity of linear models. Nevertheless, the linear model is able to identify both benchmarks with increased energy overhead (`nw-loopback` and `ipc`), regardless of their execution time overheads (`ipc` has execution time overhead but `nw-loopback` has none).

We assume that a more evenly distributed training data set overcomes the constant offset we observe and allows operators and users to conveniently identify applications particularly affected by Meltdown/Spectre mitigations. Hence, for applications with low overhead the mitigations can be enabled without disadvantages and for applications with moderate or high overheads the operators and users can decide on a per-application basis whether the higher security level or lower costs are more important.

Of course, the (relatively simple) linear model could be further refined to achieve better accuracy results. However, we argue that more complex models need (a) more training data and (b) more input features. This leads to increased efforts for training data collection and model training and thus it takes longer to compensate these efforts by applying the model and identify applications where mitigations can be disabled. At the moment, we can collect all four input features for the model with a single perf run and thus with low effort and train our model with a relatively small benchmark set. Hence, it is easy to break even with the costs of model creation quickly. Furthermore, the costs for re-training with new and updated software mitigations and other hardware platforms remain low. Therefore, more complex models' potentially better accuracy may not compensate for the increased creation efforts.

6 RELATED WORK

To the best of our knowledge, little research has been conducted on the energy overhead of Meltdown/Spectre mitigations. Loukeris briefly analysed the energy and execution overheads of Meltdown and Spectre mitigations [14]. He conducted energy measurements with a relatively low maximum sampling rate of 1 Hz. His results show that the energy overheads depend on the application properties and for the paper's benchmark set range from no noticeable overhead to up to 26 %. This finding is confirmed by our measurements. Although the paper allows a general assessment, whether energy overheads exist in general, it does not execute a fine-grained analysis, as done in this work, and its results have only limited precision due to the low sampling rate.

The performance impacts in terms of the execution time of the Meltdown/Spectre mitigations have been examined by academic [1, 17, 18] and non-academic [7] communities. Prout et al. [17] analyse the impact of Meltdown/Spectre mitigations on the performance of *high-performance computing* (HPC) workloads. They observe significant overheads for software mitigations and CPU microcode updates throughout their set of real-world applications, especially for network and I/O-intensive applications. This allows the assumption that our results can be transferred to HPC applications and vice versa. Microcode updates are not considered in this paper. However, the energy overheads for microcode updates are an interesting topic for future work.

Ren et al. [18] perform a study on the performance evolution of basic Linux kernel operations in terms of latency. Meltdown/Spectre mitigations are included in this study, but the study is not limited to these and a long-term overview for kernel versions between 3.0 and 4.20 is given. This study identifies KPTI and the retpoline patch as main factors for an increase in system call latency.

The work in [7] examines the impact of KPTI only. It identifies the system call and page fault rate as most important factors for performance regressions and confirms the previous results that the application behaviour significantly influences to which extent overheads can be observed. It proposes that overheads can be anything between 1 % and 800 %.

Alhubaiti et al. [1] examine the impact of Meltdown/Spectre mitigations on cryptographic algorithms. They observe small performance overheads, which align with our expectations as cryptographic algorithms usually do not require much operating system activity but are CPU and memory intensive.

Modelling software and applications' power demand is a well-established field of research nowadays [9, 20]. Even in the security community, the need for fine-grained power models has been identified [15]. Nacci et al. argue that (security) applications need to run with adaptive energy goals referred to as *green security*. For the development of such applications, accurate and precise power models are required. This becomes even more important as in recent years power side-channel attacks have been developed [12, 19]. Precise power models can help to identify and avoid power side channels during software development. For example, deployed in a continuous-integration toolchain, power side channels over different software versions and different workloads can be identified.

7 CONCLUSION

In this work, we provide a detailed energy overhead analysis of Meltdown and Spectre software mitigations. We show that energy overheads are highly application-specific and range between no to 72 % overhead. Thereby, the overheads differ depending on the used subsystems. Especially, subsystems including a lot of interactivity with the operating systems (i.e., system calls and context switches) show higher overheads. Furthermore, we investigate the correlation between energy and execution time overhead and show that both are often correlated. However, for some applications there are no execution time overheads, but large energy overheads. Finally, we combine the gathered information in an energy model in order to predict the energy overhead for an application. The model is capable of identifying applications with high energy overheads at low creation and execution costs.

ACKNOWLEDGEMENTS

This work was partially funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – project number 146371743 – TRR 89 "Invasive Computing", and under grant no. SCHR 603/15-2 ("eLARN"). The authors acknowledge support from the Bundesministerium für Bildung und Forschung (BMBF, Federal Ministry of Education and Research) in Germany for the project AI-NET-ANTILLAS 16KIS1315.

REFERENCES

[1] Omar Alhubaiti and El-Sayed M. El-Alfy. 2019. Impact of Spectre/Meltdown Kernel Patches on Crypto-Algorithms on Windows Platforms. In *Proceedings of the 2nd International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT'19)*. IEEE, 1–6.

[2] Luiz Barroso and Urs Hölzle. 2007. The Case for Energy-Proportional Computing. *IEEE Computer* 40, 12 (2007), 33–37.

[3] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas Wenisch, Yuval Yarom, and Raoul Strackx. 2019. Breaking Virtual Memory Protection and the SGX Ecosystem with Foreshadow. *IEEE Micro* 39, 3 (2019), 66–74.

[4] Claudio Canella, Daniel Genkin, Lukas Giner, Daniel Gruss, Moritz Lipp, Marina Minkin, Daniel Moghimi, Frank Piessens, Michael Schwarz, Berk Sunar, Jo Van Bulck, and Yuval Yarom. 2019. Fallout: Leaking Data on Meltdown-Resistant CPUs. In *Proceedings of the 26th Conference on Computer and Communications Security (CCS'19)*. ACM Press, 769–784.

[5] Sanjeev Das, Jan Werner, Manos Antonakakis, Michalis Polychronakis, and Fabian Monrose. 2019. SoK: The Challenges, Pitfalls, and Perils of Using Hardware Performance Counters for Security. In *Symposium on Security and Privacy (SP'19)*. IEEE, 20–38.

[6] Xiaobo Fan, Wolf-Dietrich Weber, and Luiz Andre Barroso. 2007. Power Provisioning for a Warehouse-Sized Computer. In *Proceedings of the 34th Annual International Symposium on Computer Architecture (ISCA'07)*. ACM, 13–23.

[7] Brendan Gregg. 2018. KPTI/KAISER Meltdown Initial Performance Regressions. http://www.brendangregg.com/blog/2018-02-09/kpti-kaiser-meltdown-performance.html Accessed 02/19/2021.

[8] Taliver Heath, Eduardo Pinheiro, Jerry Hom, Ulrich Kremer, and Ricardo Bianchini. 2002. Application Transformations for Energy and Performance-aware Device Management. In *Proceedings of the 11th International Conference on Parallel Architectures and Compilation Techniques (PACT'02)*. IEEE, 121–130.

[9] Timo Hönig, Benedict Herzog, and Wolfgang Schröder-Preikschat. 2019. Energy-demand Estimation of Embedded Devices using Deep Artificial Neural Networks. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing (SAC'19)*. ACM, 617–624.

[10] Timo Hönig, Heiko Janker, Christopher Eibel, Oliver Mihelic, and Rüdiger Kapitza. 2014. Proactive Energy-Aware Programming with PEEK. In *Proceedings of the Conference on Timely Results in Operating Systems (TRIOS'14)*. USENIX, 1–14.

[11] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. 2019. Spectre Attacks: Exploiting Speculative Execution. In *Symposium on Security and Privacy (SP'19)*. IEEE, 1–19.

[12] Moritz Lipp, Andreas Kogler, David Oswald, Michael Schwarz, Catherine Easdon, Claudio Canella, and Daniel Gruss. 2021. PLATYPUS: Software-based Power Side-Channel Attacks on x86. In *Symposium on Security and Privacy (SP'21)*. IEEE, 1–17.

[13] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. 2018. Meltdown: Reading Kernel Memory from User Space. In *Proceedings of the 27th USENIX Security Symposium*. 973–990.

[14] Michail Loukeris. 2019. Efficient Computing in a Safe Environment. In *Proceedings of the 27th Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE'19)*. ACM, 1208–1210.

[15] Alessandro Antonio Nacci, Francesco Trovò, Filippo Maggi, Matteo Ferroni, Andrea Cazzola, Donatella Sciuto, and Marco D. Santambrogio. 2013. Adaptive and Flexible Smartphone Power Modeling. *Mobile Networks and Applications* 18, 5 (2013), 600–609.

[16] David Lorge Parnas. 1979. Designing Software for Ease of Extension and Contraction. *IEEE Transactions on Software Engineering* SE-5, 2 (1979), 128–138.

[17] Andrew Prout, William Arcand, David Bestor, Bill Bergeron, Chansup Byun, Vijay Gadepally, Michael Houle, Matthew Hubbell, Michael Jones, Anna Klein, Peter Michaleas, Lauren Milechin, Julie Mullen, Antonio Rosa, Siddharth Samsi, Charles Yee, Albert Reuther, and Jeremy Kepner. 2018. Measuring the Impact of Spectre and Meltdown. In *Proceedings of the 22nd High Performance Extreme Computing Conference (HPEC'18)*. IEEE, 1–5.

[18] Xiang Ren, Kirk Rodrigues, Luyuan Chen, Camilo Vega, Michael Stumm, and Ding Yuan. 2019. An Analysis of Performance Evolution of Linux's Core Operations. In *Proceedings of the 27th Symposium on Operating Systems Principles (SOSP'19)*. ACM, 554–569.

[19] Adrian Tang, Simha Sethumadhavan, and Salvatore Stolfo. 2017. CLKSCREW: Exposing the Perils of Security-oblivious Energy Management. In *Proceedings of the 26th USENIX Security Symposium*. 1057–1074.

[20] Gene Wu, Joseph L. Greathouse, Alexander Lyashevsky, Nuwan Jayasena, and Derek Chiou. 2015. GPGPU Performance and Power Estimation using Machine Learning. In *Proceedings of the 21st International Symposium on High Performance Computer Architecture (HPCA'15)*. IEEE, 564–576.