SUIT: Secure Undervolting with Instruction Traps

Jonas Juffinger
Graz University of Technology
Graz, Austria
jonas.juffinger@iaik.tugraz.at

Stepan Kalinin
North Carolina State University
Raleigh, NC, USA
skalini@ncsu.edu

Daniel Gruss
Graz University of Technology
Graz, Austria
daniel.gruss@iaik.tugraz.at

Frank Mueller
North Carolina State University
Raleigh, NC, USA
fmuelle@ncsu.edu

Abstract

Modern CPUs dynamically scale voltage and frequency for efficiency. However, too low voltages can result in security-critical errors. Hence, vendors use a generous safety margin to avoid errors at the cost of higher energy overheads.

In this work, we present SUIT, a novel hardware-software co-design to reduce the safety margin substantially without compromising reliability or security. We observe that not all instructions are equally affected by undervolting faults and that most faultable instructions are infrequent in practice. Hence, SUIT addresses infrequent faultable instructions via two separate DVFS curves, a conservative and an efficient one. For frequent faultable instructions, SUIT statically relaxes the critical path in hardware. Consequently, the instruction is not faultable anymore on the efficient DVFS curve at the cost of performance overheads for this specific instruction. For infrequent faultable instructions, SUIT introduces a trap mechanism preventing execution on the efficient curve. With this trap mechanism, SUIT temporarily switches to the conservative DVFS curve and switches back if no faultable instruction was executed within a certain time frame. We evaluate all building blocks of SUIT, using both measurements on real hardware and simulations, showing a performance overhead of 3.79 %, and a CPU efficiency gain of 20.8 % on average on SPEC CPU2017.

ACM Reference Format:
Jonas Juffinger, Stepan Kalinin, Daniel Gruss, and Frank Mueller. 2024. SUIT: Secure Undervolting with Instruction Traps. In *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24), April 27-May 1, 2024, La Jolla, CA, USA.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3620665.3640373

ASPLOS '24, April 27-May 1, 2024, La Jolla, CA, USA

ACM ISBN 979-8-4007-0385-0/24/04
https://doi.org/10.1145/3620665.3640373

1 Introduction

With increasing energy consumption of information technology [29], energy efficiency has become a crucial design goal for modern computers. However, manufacturers use generous voltage guardbands in their CPUs causing a higher energy consumption than necessary. To counteract this, CPUs optimize their efficiency with Dynamic Voltage and Frequency Scaling (DVFS) [23, 27, 59]. DVFS allows the CPU to switch between power states with different power consumption and performance. While vendors define different voltage-frequency pairs for each power state, finding an optimal DVFS curve is non-trivial. A voltage slightly too low already makes the CPU unreliable, as a small set of instructions will produce faulty results [8, 31, 47, 56, 60]. At this point the system is vulnerable and unfit for use in production. The susceptibility to faults depends on various factors during manufacturing and operation of a chip [42, 50].

Still, reducing these voltage guardbands, i.e., undervolting a CPU, is a highly researched topic because it can greatly reduce the power consumption of CPUs [6, 7, 16, 30, 35, 36, 43, 51, 52]. While this is most relevant in mobile contexts where energy availability is limited, desktop computers and servers also benefit from lower costs for cooling and energy. Although it may seem counter-intuitive, undervolting not only increases the efficiency but it can also increase the performance of CPUs. Modern CPUs are dynamically throttled to stay within the specified thermal and power limits [17, 54, 63]. With lower energy consumption, the CPU can sustain higher clock frequencies longer while staying below these limits, increasing performance.

However, reducing these guardbands also introduces severe security issues as shown by prior work [8, 31, 47, 56, 60]. These software-based fault attacks undermine all security guarantees of a CPU, even breaking trusted execution environments, which motivates the basic questions of our work:

Can system-level mechanisms guarantee security and reliability against undervolting-induced faults? How much energy do such mechanisms cost? Can these mechanisms be lightweight enough to enable substantial energy savings?

In this work, we present **SUIT**, a novel hardware-software co-design enabling a substantial increase in CPU efficiency

without compromising reliability or security. SUIT is the first work on efficiency improvement by undervolting that builds on the observation that not all instructions are equally susceptible to undervolting faults [32, 47]. In contrast to prior work [6, 7, 16, 30, 35, 36, 43, 51, 52], not explicitly protecting the aging and temperature guardband when undervolting, SUIT keeps these essential voltage guardbands by using this difference in the voltages required by different instructions.

Besides the single DVFS curve current CPUs already have, SUIT adds a second, more efficient DVFS curve that the vendor determines by *excluding* a set of instructions faulting first when the voltage is lowered. These instructions are then disabled while SUIT uses the efficient DVFS curve. We introduce a trap mechanism to handle the execution of disabled instructions, similar to existing traps, e.g., for invalid opcodes. When the instruction stream runs into a disabled instruction, SUIT transfers control to the operating system that can either emulate the instruction in software or temporarily switch to the conservative DVFS curve where the instruction can be executed with a sufficiently high voltage. SUIT uses a deadline mechanism to determine when to switch back to the efficient DVFS curve: If no disabled instruction is executed until the deadline is reached, SUIT switches back to the efficient DVFS curve. The second DVFS curve, disabling of specific instructions, trap mechanism and deadline mechanism require hardware changes of the CPU.

Our analysis, based on execution traces of applications on x86 systems, shows that faultable instructions typically occur infrequently. On average over all SPEC CPU2017 benchmarks, one such faultable instruction is encountered every 5 billion instructions. SUIT uses dynamic DVFS curve switching or emulation for these kinds of instructions. Other faultable instructions, such as the integer multiply instruction (`IMUL`) occurring as frequently as every 560 instructions, are the exception. SUIT relaxes their critical path in hardware to lower their voltage requirement. Thus, SUIT can handle *any* frequency of faultable instructions.

The first building block of SUIT addresses the *infrequently faultable instructions*. If a disabled instruction is executed, SUIT can either dynamically switch DVFS curves or emulate the instruction. Which method is more efficient depends on the delays of DVFS curve adjustments, the number of independent DVFS domains and the distribution and frequency of disabled instructions in the workload. Due to SUIT's software component, it can dynamically switch between DVFS curve adjustments and emulation. We also evaluate the impact of SIMD instructions on performance and efficiency. With SUIT, compiling applications without SIMD instructions can increase efficiency in about half of the tested workloads.

The second central building block of SUIT addresses *frequent faultable instructions*. As frequent exceptions would hurt system performance and efficiency, we instead relax their critical path in the hardware design. This results in a small performance overhead but has the benefit that the instruction is stable at lower voltages. We evaluate different faultable instructions and find our second technique to be only relevant for the x86 integer multiplication instructions `IMUL` and `MUL`, for short `IMUL`. The performance overhead of a SUIT-adjusted, one clock cycle longer, x86 `IMUL` instruction in the SPEC CPU2017 benchmarks is only 0.03 % on average with a maximum of 1.60 % for the 525.x264 benchmark. While the concept of prolonging the critical path is investigated for `IMUL` in this work, the principle can be applied to another opcode if it were to be a frequent faultable instruction on a given architecture. This provides a path forward for SUIT, even though we did not observe any such case beyond `IMUL`.

We evaluate SUIT with measurements on different CPUs from AMD and Intel. The short delay to change the core frequency, the per-core voltage domains and the positive performance impact of undervolting make Intel Xeon CPUs preferred for SUIT. SUIT uses an optimized strategy to change frequency and voltage to increase the efficiency as much as possible while causing only a negligible performance degradation in some cases. With this strategy, SUIT increases the energy efficiency by over 12 % on average over all SPEC CPU2017 benchmarks on Intel CPUs. When compiling programs without faultable SIMD instructions they can always run on the efficient DVFS curve but have the overhead of not using SIMD. With this, SUIT increases the efficiency by 19 % but requires the recompilation of applications. In our security analysis, we show that SUIT prevents the execution of instructions in the faultable set on the efficient DVFS curve. Based on this design, we use a reductionist argument to show that SUIT's security is equivalent to that of systems today. Hence, SUIT is a viable approach for substantial energy savings in a secure manner.

Contributions. Our contributions are as follows:

- We present SUIT, a novel software-hardware co-design enabling substantial energy savings through tailored DVFS adjustments while maintaining security.
- We analyze the undervolting potential due to faultable instructions and analyze their frequency in real-world software, revealing that they are often used in short bursts, e.g., for encryption, which is ideal for SUIT.
- We measure DVFS adjustment delays on real hardware showing large differences in CPUs of different vendors.
- By evaluating SUIT, we observe no performance loss and a reduction in power consumption by 14 %, resulting in an energy efficiency gain of up to 20 %.
- We show that the security of SUIT is equivalent to the security of today's CPUs in a reductionist argument.

Outline. In Section 2, we provide background. In Section 3, we present the design of SUIT. In Section 4, we detail the hardware and software changes for SUIT. In Section 5, we determine parameters for SUIT on real hardware-software setups. In Section 6, we evaluate SUIT's performance, efficiency and security. We discuss related work in Section 7, our results in Section 8 and conclude in Section 9.

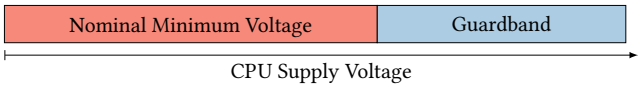

Figure 1. Typical supply voltage of a CPU. It must be higher than the nominal minimum voltage to account for all uncertainties of the CMOS circuit, aging, temperature and its surroundings.

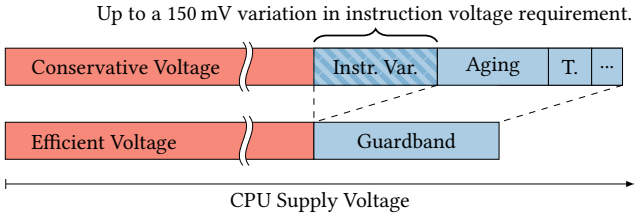

Figure 2. SUIT does not eliminate the aging or temperature (T) guardband. Instead, it utilizes the variation in voltage requirements of different instructions, especially IMUL and SIMD instructions.

2 Background

We provide background on CMOS circuits, voltage guardbands, voltage-related faults and DVFS.

2.1 Power Consumption of CMOS Circuits

The dynamic power consumption of a CMOS circuit, $P_{dyn} = C_L \cdot (V_{DD})^2 \cdot f_{CLK}$, depends on the load capacitance C_L, supply voltage V_{DD} and clock frequency f_{CLK} [44]. CMOS circuits mainly consume power when switching [44], as on every switch a CMOS gate must charge or discharge its C_L. The switching energy depends on the supply voltage V_{DD} squared [44]. This relation is also the basis for many power consumption side-channel attacks [33, 39, 41, 57, 63].

Supply voltage V_{DD} and clock frequency f_{CLK} also depend on each other [17]. In a CMOS circuit many transistors implement combinatorial logic with a propagation delay t_P, depending on V_{DD} [16, 49]. The block with the longest t_P is the *critical path* of the circuit and determines the maximum clock frequency $f_{CLK_m} = t_{crit}(V_{DD})^{-1}$. If the clock frequency is too high, data arrives late, manifesting in malfunctioning of the circuit, e.g., erroneous computed values [16, 49].

2.2 Voltage Guardbands

The dependency between voltage and frequency in CMOS circuits is influenced by many factors. Process variation influences every single transistor's performance and voltage requirement. Over time, voltage requirements increase due to high temperature, age, and voltage spikes delivered by the power supply unit [24]. To counter these effects, manufacturers use a voltage guardband. The circuit is supplied with a higher voltage than the nominal minimal voltage to take these uncertainties into account as shown in Figure 1.

Aging effects like bias temperature instability and hot-carrier injection cause the threshold voltage and consequently the propagation delay to increase over time [58]. The degree of aging must be predicted accurately to design a guardband. If it is too low the device can break within its expected lifetime, if it is too large more power than necessary is used.

The propagation delay of modern sub 20 nm FinFET transistors degrades by approximately 15 % over a time span of 10 years at >100 °C [19, 46, 53, 61, 66, 67]. After ten years, the CPU must either run with a 13 % lower frequency at the same supply voltage or with a supply voltage that supports a 15 % higher frequency at age zero. Based on these numbers and real hardware measurements, we evaluate the aging and temperate guardband in Section 5.6 and 5.7. On current CPUs the aging guardband is approximately 12 % and the temperature guardband 3.5 % of the CPU supply voltage.

Table 1. Undervolting-induced instruction faults observed by Kogler et al. [32]. If an instruction faults on a specific CPU core at a specific frequency and voltage offset it counts as one fault for the number of faults. The rarely faulting instructions (right) occur on average at lower voltages than the more frequently faulting ones.

Instruction	IMUL	VOR*	AESENC	VXOR*	VANDN*	VAND*	VSQRTPD	VPCLMULQDQ	VPSRAD	VPCMP*	VPMAX*	VPADDQ
Number of Faults	79	47	40	40	30	28	24	16	9	5	3	1

2.3 Variation in Voltage Requirements

Modern CPUs are highly complex systems where vendors have multiple dials to fine-tune for the efficiency or performance optimum. In particular, they can increase the number of cycles allocated to an operation or adjust the clock frequency to account for the propagation delay within a specific part of the circuit [49]. Instructions with a long propagation delay t_P can be split up into pipeline stages, dividing t_P by the number of stages and enabling higher clock frequencies [49]. This increases the instruction's latency but keeps the throughput unchanged. An example for this is the IMUL instruction. On Intel and AMD CPUs, IMUL takes 3 clock cycles with a throughput of 1 instruction per clock cycle [15].

Even with very good optimizations some instructions can still have a significantly longer t_P than most other combinatorial acyclic logic blocks inside the CPU. This was first discovered by Murdoch et al. [47] for IMUL on Intel CPUs. In their experiments they measured that IMUL starts to produce faulty results very infrequently on one of their tested CPUs at a −100 mV undervolt level. However, apart from IMUL (they did not test SIMD instructions), the CPU was running stable down to a −250 mV level, resulting in a variation in voltage requirements of 150 mV as shown in Figure 2.

Kogler et al. [32] performed a more detailed study on this phenomenon by building a framework that automatically tests instructions for their undervolting behavior. They confirmed that across several CPUs data errors occur earlier than control-logic errors, indicating that the critical paths

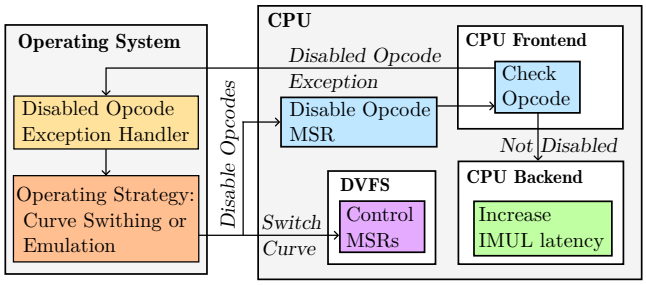

Figure 3. The overall SUIT design changes several parts of the system (colored boxes). We disable certain instructions while switching to an efficient DVFS curve and increase the latency of others. Executing disabled instructions raises a #DO exception, handled by the OS through emulation or switching of the DVFS curve.

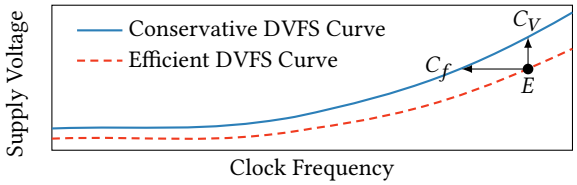

Figure 4. Switching from a p-state on the efficient DVFS curve to the conservative DVFS curve can be done in two ways: by changing the voltage (C_V) or the frequency (C_f).

in modern CPUs are more often on the data paths within instructions. The highest variation in instruction voltage they measured was more than 60 mV. Following IMUL in sensitivity to voltage levels are VOR and other SIMD instructions, as well as AESENC, as shown in Table 1.

2.4 Dynamic Voltage and Frequency Scaling (DVFS)

Based on the workload, the supply voltage and clock frequency of modern CPUs is adjusted to trade off performance and power consumption. For DVFS, the vendor defines p-states, pairs of clock frequencies and voltages including a guardband that, in combination, form a *DVFS curve*. Although the p-state is hardware-controlled, the OS can still intervene in the frequency voltage pairs on some Intel CPUs with the undocumented MSR 0x150 [45].

3 Design Overview and Implementation

In this section, we provide an overview of SUIT's design (see Figure 3) showing, on a high level, how it enables efficiency gains without affecting the system's security. With its generic design, SUIT can be integrated into different CPU architectures. For our proof-of-concept implementation and evaluation, we focus on the changes for x86-64.

SUIT provides software interfaces to disable instructions and to switch between DVFS curves. The OS disables all instructions that may fault on the efficient DVFS curve. Hence, the OS can switch to the efficient DVFS curve without affecting the system's security with faultable instructions.

3.1 SUIT's Undervolting Potential

SUIT's undervolting potential comes from two sources. First, the variation in the required voltage by different instructions, i.e., faultable instructions. Deactivating instructions that produce faulty results undetectably while the CPU continues running eliminates a security risk exploited by various attacks [8, 31, 47, 56, 60]. Second, due to the elimination of this vulnerability, a small fraction of the aging guardband can be used for undervolting limited to a certain amount of time.

The studies of Murdoch et al. [47] and Kogler et al. [32] show large differences in instruction voltage requirements between CPUs and even CPU cores due to process variation. The average over all CPUs covered in these two studies that exhibit this variation in instruction voltage requirements is 70 mV ($n = 6$, $\sigma = 44$ mV), the maximum is 150 mV. CPUs from Intel 6th generation did not exhibit this variation [32].

Data centers and cloud providers, representing a growing share of the global electric power consumption [34], procure new CPUs after a few years. For example, AWS recently increased the life span of their servers from four to five years [14]. Xeon CPUs are typically supported for less than 10 years [28]. Additionally the aging degradation is larger at higher temperatures [40, 65]. During the limited life span and with well controlled core temperatures the full aging guardband designed for the worst case is not required. Also typical consumer devices are not running continuously although the guardband is designed for nonstop use. Exploiting some fraction of the aging guardband with SUIT in the first few years can have a large impact on data center and consumer device power consumption without impact on reliability.

In Section 6, we evaluate a conservative undervolting margin of −70 mV highlighting the differences in instruction voltage requirements and with an additional 20 % of the on average 137 mV aging guardband for a combined −97 mV offset. Thus, in contrast to previous work [7, 16, 30, 35, 36, 43, 51, 52], SUIT uses either none or only a small fraction of the aging voltage guardband for undervolting.

3.2 Two DVFS Curves

A CPU with SUIT has a conservative and an efficient DVFS curve. The conservative curve is the same as on current CPUs. The efficient curve is determined by excluding the small number of instructions that fault in the "instruction voltage variation" range of the voltage supply (see Figure 2).

We introduce a new MSR that allows the operating system to select which DVFS curve to use. The CPU ensures that the efficient curve can only be used if the faultable instructions are disabled. Switching from the efficient to the conservative DVFS curve can be done in two ways, as seen in Figure 4. Section 4.3 details the different effects of the switching method on the efficiency and performance of SUIT.

3.3 Disabling Instructions

We propose a new model-specific register (MSR) that allows the operating system to disable all faultable instructions defined by the manufacturer per voltage or frequency domain. We use a reserved interrupt vector number [26] for a new *Disabled Opcode* (#DO) CPU exception. When the CPU encounters a disabled instruction it raises a #DO exception. Like other CPU exceptions, #DO preserves the current register set so that the program can continue after the exception is handled. The OS handles this new exception by following the approaches presented in Section 4.

3.4 Instruction Emulation

Additionally to switching the DVFS curve to execute faultable instructions, a system with SUIT can also emulate instructions. SUIT emulates instructions like VOR or VPCMP with non-vectorized alternatives, and AESENC with a side-channel-resilient bit-sliced AES implementation. To perform this emulation, the operating system maps emulation code into the memory of the user space program. The exception handler returns to this emulation code, which is then run in user space. After the emulation, the program returns to the kernel through, e.g., a system call, where the original register contents are restored and the program continues execution after the disabled instruction. The two transitions into the kernel and back dominate the overhead.

3.5 Security

SUIT's security is established exactly as for current CPUs, namely by the vendor determining safe DVFS curves. Today, when an instruction produces erroneous results during this process, the vendor either accepts this faultable instruction as a critical path for the CPU, or increases the instruction latency to reach a slightly more efficient DVFS curve. SUIT provides a third option. Instead of increasing the latency, with SUIT, vendors determine a second DVFS curve, which excludes the small number of faultable instructions, thereby resulting in a considerably more efficient DVFS curve. As this follows the same process as before, just with fewer instructions, we obtain the exact same security level for this slightly reduced instruction set. The details of the security evaluation are given in Section 6.9.

4 SUIT Hardware-Software Interaction

SUIT uses two building blocks for low-frequency and high-frequency faultable instructions, which are generically applicable to any CPU and microarchitecture. While both introduce overheads, we show in Section 6.3 that, overall, SUIT not only compensates these overheads but yields a significant increase in system efficiency. We assume that the vendor has determined the two DVFS curves with and without the faultable instructions during manufacturing, one of which being the DVFS curve the vendor determines already today.

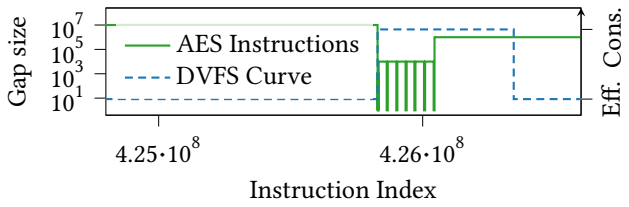

Figure 5. Detailed view of an AES instruction burst and the resulting change of the DVFS curve to conservative and back. Horizontal segments correspond to the time without AES instructions executed, with the height of the segment showing the gap size, the log_{10} of the segment's length. Vertical segments show instruction bursts.

4.1 Low-Frequency Faulting Instructions

SUIT's central idea is that the CPU generally runs on the efficient DVFS curve. SUIT handles the corner case of disabled instructions by triggering an OS handler that temporarily switches to the conservative and less efficient curve, which is secure for all instructions as shown in Figure 5. When continuing on the conservative DVFS curve after the switch, the CPU re-executes the instruction that was formerly disabled.

Our analysis shows that programs often use faultable instructions in short bursts. Based on this observation, SUIT uses a deadline mechanism to determine when to switch back to the efficient DVFS curve. Initialized with the deadline it counts down with constant speed. At zero, an interrupt is triggered to switch back to the efficient DVFS curve. Whenever the CPU executes an instruction that would be disabled on the efficient DVFS curve, the timer is reset and starts counting down again. Therefore, SUIT automatically adjusts to many frequencies of faultable instructions and avoids thrashing-like effects in many cases, where SUIT would frequently switch between the DVFS curves.

Instruction Emulation. For single instructions, emulation is faster than switching DVFS curves (see Section 6.6). For 65 % of our tested applications emulation is beneficial.

Overhead. The dynamic building block of SUIT incurs overheads from two sources. The first is due to the #DO exception and corresponding OS handler. We measured this delay to be 0.34 µs in Section 5.2. The second overhead is due to the DVFS curve switching delay or instruction emulation. SUIT only has to delay execution when switching from the efficient to the conservative curve; in the other direction, it disables the instructions and does not need to wait until the efficient curve is reached. We measured this delay on current Intel and AMD CPUs in Section 5.2. On an Intel Xeon Silver 4208, it takes on average 31 µs to change the core frequency and 335 µs to change the core voltage. On an AMD Ryzen 7 7700X it takes on average 668 µs to change the frequency.

4.2 High-Frequency Faulting Instructions

The latency of an instruction is a direct result of the propagation delay of the instruction's critical path. If the critical path

of an instruction is larger than the inverse of the required clock frequency, the instruction's critical path has to be split up into multiple stages, forming a pipeline for the instruction. The number of stages corresponds to the latency of the instruction. By increasing the number of pipeline stages and, therefore latency, the shorter critical path of the individual stages allow for lower voltages or higher frequencies [49].

The IMUL instruction family, which multiplies two integers, is quite commonly used and considerably more complex than most other ALU instructions. It is typically split up into 3 stages on various modern CPUs [2, 3, 15]. These 3 clock cycles are still very tight, limiting the vendor in further optimizing the DVFS curve due to the critical path within the IMUL instruction. Kogler et al. [32] practically confirmed this by undervolting and finding that the IMUL instruction faulted first in 91.2 % of cases. Our analysis of the instruction usage of applications in Section 5.1 shows that IMUL is the only instruction used so frequently that SUIT would permanently run on the conservative DVFS curve, preventing any potential efficiency gain. Therefore, SUIT has a second building block for frequent instructions, e.g., IMUL.

The central idea is to address this problem by statically removing frequent instructions from the set of faultable instructions at the cost of performance. We analyze the sensitivity of performance to latency increases of IMUL by adjusting its implementation in a microarchitecture simulator. It is important to note that IMUL is fully pipelined. While the latency is 3 cycles, already after the first cycle, another input can be pushed into the IMUL pipeline. Hence, changing the latency has no effect on the throughput. Overall, increasing the latency of IMUL by 1 clock cycle causes a performance overhead of 0.03 % ($n = 16$, $\sigma_{\bar{x}} = 0.15$) on average and 1.60 % ($n = 9$, $\sigma_{\bar{x}} = 0.55$) in the worst case (see Section 6.1).

Static Latency Increase is not Suitable for All Instructions. While it seems tempting to just increase the latency of all faultable instructions, this decreases performances and removes the flexibility of SUIT. As our evaluation shows, SUIT increases the efficiency on average and for many workloads significantly. Additionally, it does not decrease the performance of highly specialized workloads that use many faultable instructions (e.g., 520.omnetpp, Section 6.3). For these specialized workloads, SUIT will stay on the conservative DVFS curve, as faultable instructions occur frequently. Consequently, for specialized workloads, SUIT maintains a high performance while increasing the efficiency for others.

4.3 Operating Strategy

The operating strategy describes how the operating system controls the SUIT hardware. There are four main ways based on the two DVFS curve switching methods, see Figure 4.

Emulation. The DVFS curve is not switched but the instruction is emulated in the #DO exception handler. Therefore, every disabled instruction incurs the overhead of the exception and the emulation. Emulation is not possible for applications running in trusted execution environments.

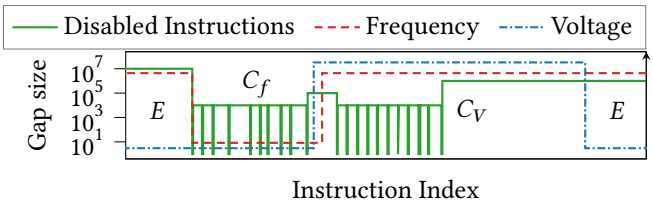

Figure 6. Detailed view of a long burst of faultable instructions and how frequency and voltage change with the fV operating strategy.

Frequency $E \leftrightarrow C_f$. Switching the DVFS curve by changing the frequency enables fast switching and is highly efficient because the CPU always stays at the lower voltage. However, the performance is lowered when running on C_f.

Voltage $E \leftrightarrow C_V$. The DVFS curve switch by voltage is approximately a magnitude slower than by frequency, resulting in a high performance impact. Nonetheless, performance is high when running on the conservative DVFS curve.

Combination (fV) $E \leftrightarrow C_f \rightarrow C_V \rightarrow E$. Changing the frequency *and* voltage can lead to an optimal balance between efficiency and performance. On a #DO exception, a quick switch to the conservative curve is realized by changing the frequency. At the same time, a voltage increase is requested. The program continues running at C_f. If the burst of potentially faulting instructions is short, the operating strategy returns to E and cancels to voltage change. If the burst is long enough for the voltage to change, the CPU runs at C_V with full performance. This incurs another frequency change stall delay. The whole sequence is depicted in Figure 6.

Trashing Prevention. If the gap between disabled instructions is bit longer than the deadline, the CPU constantly switches DVFS curves adding considerable overhead. The OS detects this by counting the number of #DO exceptions during a specified time span and increases the deadline. This ensures that the CPU stays in the conservative DVFS curve.

Parameters. The fV operating strategy and trashing prevention are optimized with four parameters: (1) The deadline (p_dl) denoting the maximum time between two potentially faulting instruction before switching back to the efficient curve; (2) the time span (p_ts) over which trashing prevention looks back and counts #DO exceptions; (3) the maximum #DO exceptions count (p_ec) in p_ts; and, if trashing is detected, (4) the deadline factor (p_df) by which the deadline is multiplied for this stable period.

A pseudo code implementing the fV operating strategy and trashing prevention for the evaluation simulation that use the parameters is shown in Listing 1.

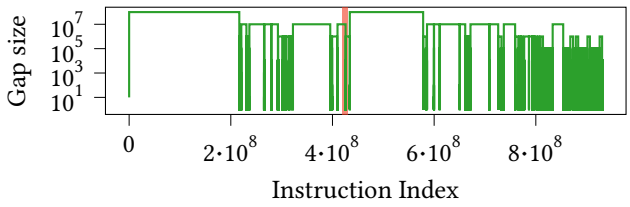

Figure 7. Timeline of AES instruction execution while VLC is streaming a 1080p video to visualize how these instructions are often executed in bursts. The timeline is truncated to the first 0.5 %. For the evaluation we take all faultable instructions into account.

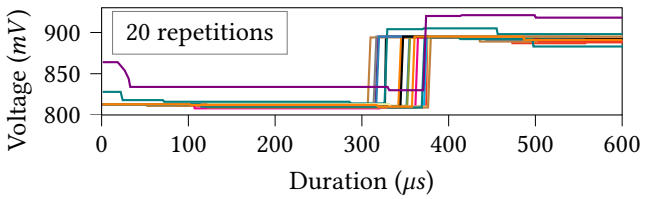

Figure 8. The core voltage of an Intel Core i9-9900K after resetting the negative voltage offset to 0 mV at time 0. Faultable instructions must not be executed until the core voltage is actually increased.

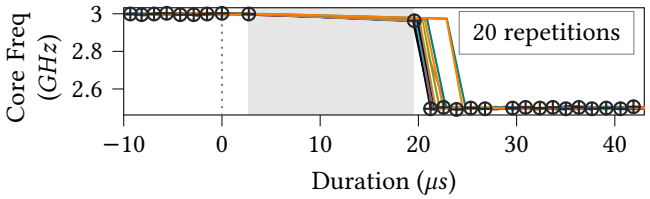

Figure 9. There is a delay between requesting a frequency change and the actual change on Intel CPUs (here an i9-9900K). Measurement samples (⊕) are shown relative to time 0 (the dotted line) where the new frequency was set. The gray area marks the duration during which the CPU stalls and no samples were measured.

5 Evaluation of Building Blocks on Real Hardware and Software

For each building block of SUIT, we perform measurements of real software and hardware to obtain realistic data for the evaluation. In Section 5.1, we analyze applications for their usage of faultable instructions. In Section 5.2, we measure the delays incurred by exceptions and changing the core voltage and frequency. To determine the effects of SUIT on efficiency, we measure how CPUs behave when running them undervolted in Section 5.4. In Section 5.5, we measure the DVFS curve of a contemporary CPU for our security evaluation. In Section 5.6, we measure the aging guardband and in Section 5.7 the temperature guardband. In Section 5.8, we measure the impact of disabling SIMD instructions on SPEC CPU2017. Subsequently, in Section 6, we use these results to evaluate the entire SUIT system.

5.1 Frequency of Disabled Instructions in Real Code

We analyze 25 applications for their usage of faultable instructions. The data collection was done using QEMU [9].

QEMU Plugin. We implement a QEMU plugin allowing us to log in detail when specific instructions are executed. We use Linux's `isolcpus` parameter to isolate one core for our application in QEMU, and use canary instructions to make sure QEMU doesn't record instructions executed outside of the workload being tested. The plugin records the trace based on instruction count. Due to modern CPUs being superscalar and instructions taking a variable number of clock cycles, we use the `INSTRUCTIONS_RETIRED` performance counter to estimate the number of instructions executed per clock cycle. We use it to convert between instruction count and CPU clock cycles in the evaluation simulation to obtain correct timings, e.g., for the deadline mechanism.

Instruction patterns. Using QEMU, we collect the data on AES and SIMD instructions from Table 1 of a client network application (nginx and VLC streaming), as well as all benchmarks in the SPEC CPU2017 benchmark suite.

The timeline shown in Figure 7 allows us to understand the distribution of AES instructions during the execution of VLC. More specifically, we can observe that AES instructions occur in bursts with gaps between them. We observed similar patterns with SIMD instructions in SPEC CPU2017 benchmarks, with larger gaps between faultable instructions accounting for larger parts of the overall execution time.

5.2 Core Voltage and Frequency Change Delays

We microbenchmark the overhead for each building block of SUIT on real systems for the simulation of SUIT in Section 6.

Voltage Change Delay. Figure 8 shows the delay to change an Intel Core i9-9900K's core voltage over 20 repetitions. We use MSR `0x150` to set voltage offsets and `MSR_IA32_PERF_STATUS` to read the current CPU core voltage. In a kernel module, we first decrease the voltage with a negative offset and wait for it to manifest. Afterward, we raise the voltage to its original level (time 0 in the figure) and measure the time it takes for the voltage to stabilize by polling `MSR_IA32_PERF_STATUS`. As shown in Figure 8, it takes 350 µs ($n = 20$, $\sigma = 22$) on average, with a maximum of 379 µs.

Frequency Change Delay. Figure 9 shows the delay on an Intel Core i9-9900k to change the clock frequency over 20 repetitions. We measure with a kernel module and the `APERF` and `MPERF` counters to get the current CPU frequency [27].

On the i9-9900K, we write `MSR_IA32_PERF_CTL` to change the frequency, which takes on average 22 µs ($n = 20, \sigma = 0.21$) with a maximum of 24.8 µs. We also confirm that there is one frequency domain. After setting the frequency, all cores of the i9-9900K stall illustrated as a gray area in Figure 9, where no samples were taken. The first sample after the stall

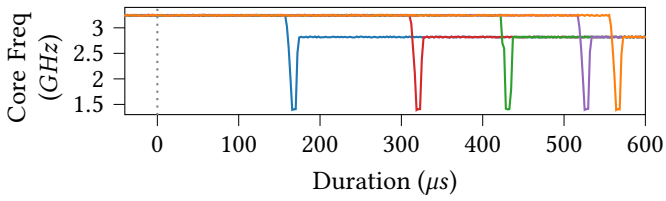

Figure 10. The delay between requesting a frequency change and the change of the frequency on an AMD Ryzen 7 7700X with per-core frequency domains. The CPU core does not stall.

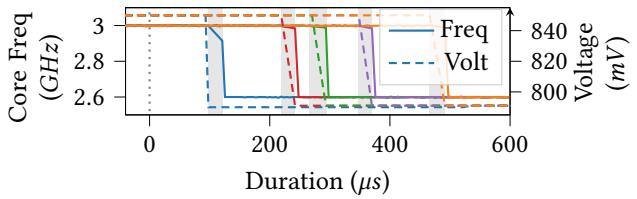

Figure 11. The delay between requesting a frequency change and the change of the voltage and frequency on an Intel Xeon Silver 4208 with per-core frequency and voltage domains.

still shows a high frequency, while it is actually already low, due to the APERF frequency being updated late during the stall. We verified this by reading the p-state value after the stall. The behavior is equal when increasing the frequency.

Figure 10 shows 5 frequency changes on the AMD Ryzen 7 7700X using `cpufreq_set`. The frequency change takes 668 µs (n = 10, σ = 292) on average. The core does not stall.

Per-Core Voltage and Frequency Change Delay. Figure 11 shows 5 p-state changes on an Intel Xeon Silver 4208 when changing the frequency with `MSR_IA32_PERF_CTL`. Intel Xeon CPUs since Haswell-EP have per-core voltage *and* frequency domains (PCPS) [21]. But they are coupled and always move in tandem. When changing the p-state the CPU core always first changes the voltage and then the frequency, regardless of the direction of the change. The frequency change looks similar to the one of the i9-9900K, suggesting that the same clock source is used. The voltage change takes on average 335 µs (n = 98, $\sigma_{\bar{x}}$ = 135) and the following frequency change 31 µs (n = 98, $\sigma_{\bar{x}}$ = 2.3) during which the CPU core stalls for 27 µs (n = 98, $\sigma_{\bar{x}}$ = 2.5).

5.3 Exception and Emulation Call Delay

We measure the end-to-end delay from a CPU exception to the invocation of the corresponding handler in the Linux kernel and the return back to user space.

Exception Delay. We use the *invalid opcode* exception, which closely resembles our *disabled opcode* exception. In user space, we store the current Time Stamp Counter (TSC) value in a register right before executing `UD2` to generate an invalid opcode. The exception is handled by our modified Linux kernel, which stores the current TSC value at the beginning of the exception handler. The difference of the two TSC values is the delay for entering the exception handler. This delay is 0.34 µs (n = 10, $\sigma_{\bar{x}}$ = 0.04) on an Intel i9-9900K and 0.11 µs (n = 10, $\sigma_{\bar{x}}$ = 0.02) on an AMD 7700X.

Emulation Call Delay. To emulate instructions in user space the kernel is entered twice: First on the *invalid opcode* exception from where it returns to the emulation code. Afterward from the emulation code back into the kernel to continue normal program execution. We measure this delay similarly to the exception delay with the `UD2` instruction. However, the exception handler returns to another function in user space that executes a second `UD2` from where the kernel jumps back to after the initial `UD2`. We read the TSC counter before and after the first UD2 instruction. This delay is 0.77 µs (n = 10, $\sigma_{\bar{x}}$ = 0.14) on average on an Intel i9-9900K and 0.27 µs (n = 10, $\sigma_{\bar{x}}$ = 0.02) on an AMD 7700X.

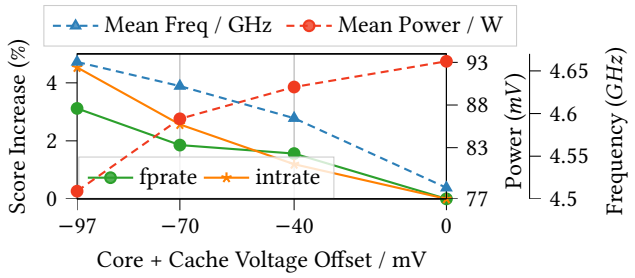

Figure 12. Performance increase for multiple undervolting offsets in SPEC CPU2017 on an Intel i9-9900K. With a voltage offset of −97 mV the SPEC CPU2017 score increases by 3.8 % and the average power consumption decreases by 16 %. The second and third y-axis show the power in W and frequency in GHz.

Table 2. Average performance (SPEC CPU2017 score) increase and power savings for different CPUs.
*AMD CPUs were undervolted using AMD's Curve Optimizer.

CPU	V_{off}	Score	Power	Freq.	Eff.
i5-1035G1	−70 mV	+6.0 %	−0.1 %	+8.5 %	+6.1 %
	−97 mV	+7.9 %	−0.5 %	+12 %	+8.4 %
i9-9900K	−70 mV	+2.2 %	−7.2 %	+2.6 %	+10 %
	−97 mV	+3.8 %	−16 %	+3.3 %	+23 %
7700X*	−70 mV	+1.4 %	−9.8 %	+1.8 %	+12 %
	−97 mV	+1.9 %	−15 %	+1.8 %	+20 %

5.4 Efficiency and Performance of Undervolting

The steady state performance of most CPUs is limited by their thermal design power (TDP), which must not be exceeded for longer periods. Hence, decreasing the CPU core voltage decreases power consumption, which subsequently allows frequencies to be increased, resulting in higher performance.

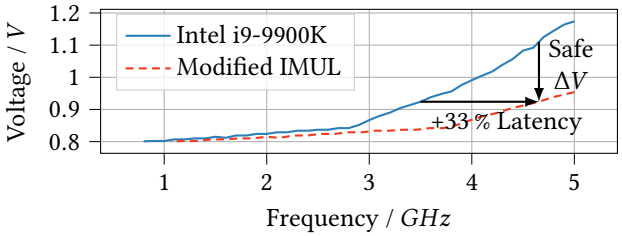

Figure 13. The stable frequency-voltage pairs on an Intel Core i9-9900K measured by fixing the frequency and reading the voltage with MSR `0x198`. The modified `IMUL` plot shows the safe voltages for `IMUL` when increasing its latency from 3 to 4 clock cycles.

Table 3. Clock frequency and fan RPM to run the i9-9900K at a specific temperature and the resulting maximum undervolting offset.

f_{CLK}	Fan RPM	t_{core}	V_{off}
4 GHz	1800 (max)	50 °C	−90 mV
4 GHz	300	88 °C	−55 mV

Table 4. The performance impact of disabling SSE and AVX on SPEC CPU2017. All benchmarks exceeding 5 % impact are shown.

	fprate	intrate	508	521	538	554	525	548
i9-9900K	−4.1 %	0.5 %	−22 %	−1.4 %	−12 %	−3.3 %	7.0 %	7.7 %
7700X	−5.9 %	2.6 %	−35 %	−5.3 %	−9.0 %	−19 %	22 %	6.8 %

We measure the CPU package power with Intel's and AMD's Running Average Power Limit (RAPL) [1, 4, 27] interfaces while undervolting the CPU to get realistic values for the power reduction. The accuracy of RAPL is sufficient for this purpose [20]. During this measurement we run the CPUs outside the vendor's defined power states down to voltage offset levels close to where the first faults happen. This is possible due to the aging and temperature guardband. Without SUIT, this is *not* recommended and can have a severe impact on system security [8, 31, 47, 56, 60]. With SUIT, undervolting is possible due to the variation in voltage required by different instructions.

Figure 12 shows the score of the SPEC CPU2017 benchmarks, power consumption and core frequency with decreasing CPU core voltage offsets on an Intel Core i9-9900K. Table 2 shows the results of three CPUs. The efficiency change is computed by one over the change in benchmark duration (score) multiplied with the change in power consumption. If a CPU finishes the benchmark in half the time (score +100 %) while using half as much energy (power −50 %) the efficiency increases by $(0.5 \cdot 0.5)^{-1} = 400\,\%$.

The i5-1035G1 seems to be limited by the TDP, with the power consumption changing negligible but the frequency increasing significantly. We undervolted the AMD CPUs with the curve optimizer [11] in the BIOS as it does not have a MSR `0x150` like Intel CPUs. The curve optimizer value translates to an undervolt offset by multiplying it by 3 – 5 mV. We selected values of -18 and -24. Intel does not allow undervolting the Xeon Silver 4208 CPU at the moment.

5.5 Frequency-Voltage Pairs

For the security analysis in Section 6.9, we measure the p-states of an Intel Core i9-9900K CPU. Figure 13 shows the pairs for guaranteed stable operation predefined by the vendor. The modified `IMUL` plot shows the safe voltage for `IMUL` with a 4 clock cycle latency, see Section 6.9.

5.6 Aging Guardband

The propagation delay of modern sub 20 nm FinFET circuits degrades by approximately 15 % over a time span of 10 years [19, 46, 53, 61, 66, 67]. To counter aging without decreasing the frequency, the voltage guardband at the beginning of the life cycle must support a 15 % higher frequency than the maximum frequency of the CPU. This ensures that the voltage is high enough after the propagation delay increased by 15 % after 10 years due to aging.

Using the p-states of our Intel Core i9-9900K from Figure 13 we can calculate the size of the aging guardband for this CPU. At 5 GHz the CPU core voltage is 1.174 V. The gradient from 4 GHz to 5 GHz is 183 $^{\text{mV}}/_{\text{GHz}}$. This results in an aging voltage guardband of $5\,\text{GHz} \cdot 15\,\% \cdot 183\,^{\text{mV}}/_{\text{GHz}} = 137\,\text{mV}$ or 12 %. This is in line with previous work [36], where authors were able to undervolt their new CPUs by approximately 100 mV more than we are able to on our aged CPUs.

5.7 Temperature Guardband

To estimate the size of the temperature guardband we measure the maximum undervolting offset at different core temperatures on an Intel Core i9-9900K. We influence the core temperature by adjusting the CPU fan speed. With a low fan speed the CPU always clocks as highly as possible under thermal throttling constraints, which must not exceed 90 °C. On average over all SPEC CPU2017 benchmarks, we run at 88 °C since a few benchmarks never reach 90 °C. Table 3 shows the maximum undervolting offset at 88 °C and 50 °C. 50 °C is twice the room temperature the CPU was running in, making it a realistic lower bound temperature for a CPU under full load. The difference in voltage requirement is 35 mV, which is 3.5 % of the 991 mV core supply voltage at 4 GHz.

5.8 SPEC CPU2017 Without SIMD instructions

A program compiled without SSE and AVX support does not execute any SIMD instructions. All instruction in Table 1 except `IMUL` and `AESENC` are SIMD instructions. If the performance overhead is smaller then the gain of SUIT, these programs can be executed without ever causing a `#DO` exception because `IMUL` is hardened in CPUs with SUIT. We measure the influence of SIMD on SPEC CPU2017 and compare these numbers in Section 6.7. Table 4 shows the impact on the benchmark score for the floating point (FP) and integer

Table 5. The gem5 system used for instruction latency evaluation.

CPU	x86-64, 2 Core, 3 GHz, O3 (Out-Of-Order) CPU
DRAM	2 Channel, 3 GB DDR4_2400_8x8
Cache	64 kB L1I, 32 kB L1D, 2 MB LLC
gem5 Mode	Full System
OS	Ubuntu 20.04.1 with Linux kernel v5.19.0

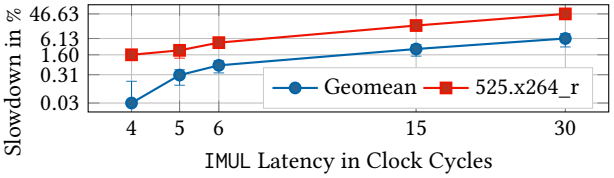

Figure 14. Slowdown in % with increasing latency of IMUL. For small increases the relationship does not grow linearly, we suspect that the out-of-order execution hides the additional latency.

suite and for selected benchmarks. The FP score decreases on both tested CPUs as expected, while the integer score *increases*. We suspect AVX throttling being the root cause for the increase [12, 37].

6 Evaluation

We evaluate the effects of SUIT on efficiency and performance on a variety of benchmarks and different CPUs.

6.1 Increased IMUL Latency

We use the gem5 simulator to evaluate the performance impact of increasing the latency of IMUL by 1 clock cycle to 4 clock cycles. The configuration is shown in Table 5. We ran the SPEC CPU2017 benchmarks in gem5 with the SPECcast tool by Prieto et al. [55]. It only runs representative parts with the out-of-order CPU model that takes longer to simulate. We also implemented the MSR to disable instructions and #DO exception and modified Linux v5.19 to handle the exception.

As shown in Figure 14, an IMUL latency of 4 clock cycles causes a 0.03 % ($n = 8$, $\sigma_{\bar{x}} = 0.15$) slowdown on average over all SPEC CPU2017 benchmarks. Frequent IMUL instructions slow down 525.x264_r most (by 1.60 % ($n = 9$, $\sigma_{\bar{x}} = 0.55$)). 0.99 % of all instructions executed by 525.x264_r were IMUL compared to 0.07 % on average over all other benchmarks.

Figure 14 also shows the slowdown for larger IMUL latencies. Small increments have only a low impact on performance because they are hidden in the out-of-order execution of CPUs. Latencies of 4 and 5 clock cycles are more easily absorbed. With higher latencies, we can see an almost linear relationship between IMUL latency and slowdown.

6.2 Instruction Trace-Based Evaluation

We use the data from Section 5 to evaluate the efficiency and performance impact of SUIT for a range of applications.

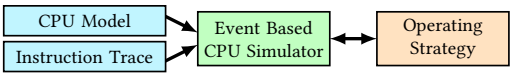

Figure 15. The components of our instruction trace-based simulator.

```python
class Operating_Strategy_fV:
  def disabled_instruction_exception_handler():
    # we wait for the frequency to change
    cpu.change_pstate_wait(DVFS.Cf)
    # and request the voltage change
    cpu.change_pstate_async(DVFS.Cv)

    cpu.set_instructions_disabled(False)

    # trashing prevention
    if exception_count_in_timespan(p_ts) >= p_ec:
      cpu.set_timer_interrupt(p_dl * p_df)
    else:
      cpu.set_timer_interrupt(p_dl)

  def timer_interrupt_handler():
    cpu.set_instructions_disabled(True)
    cpu.change_pstate_async(DVFS.E)
```

Listing 1. Implementation of the fV operating strategy (Section 4.3) with trashing protection. On a disabled opcode exception, the CPU switches to the *conservative* DVFS curve by changing the frequency (C_f), a voltage change (C_V) is started and all instructions are enabled. The trashing prevention sets a count-down timer that will switch the CPU to the *efficient* DVFS curve after expiration – unless a faultable instruction was executed prior to timer expiration, which would have reset the count-down timer to the deadline.

Methodology. Our event-based simulator models a CPU executing an instruction stream from Section 5.1 and an operating strategy from Section 4.3 as shown in Figure 15. The operating strategy is defined as a class that communicates with the simulated CPU as shown in Listing 1. The instructions in Table 1 are disabled on the efficient DVFS curve. The simulated CPU behaves as given by the base measurements from Section 5. The efficient DVFS curve causes changes in the power consumption and performance at different voltage offsets as shown in Section 5.4: −70 mV from the variation in instruction voltage requirements and −97 mV with an additional 20 % of the aging guardband, see Section 3.1.

Transitioning between these curves slows down the simulation by the delays given in Section 5.2. As the latency of IMUL is increased in all CPUs, we also factor in its negative performance impact from Section 6.1 for this evaluation.

To simulate multi-core CPUs with a single frequency domain, we record multiple instruction streams. During the simulation, one instruction stream is pinned on one core. A DVFS curve change subsequently impacts *all* cores.

Instruction Emulation. To estimate the overhead of instruction emulation, we add the overhead from each SPEC benchmark compiled without SIMD instructions from Section 5.8. Additionally, we add the emulation call delay from Section 5.3 to any execution of disabled instructions.

Applications. We record instruction traces of a wide range of applications. The first set comprises all 23 benchmarks of SPEC CPU2017 [10]. Since they do not contain network workloads apart from the simulated network by `520.omnetpp`, we also test Nginx [25] (100 kB files over an HTTPS connection with AES encryptions and decryptions), using the `wrk` benchmarking tool [64]. The network is a virtual network between the QEMU virtual machine running Nginx and its host running wrk. To benchmark the network client side, we use VLC [62] streaming a 1080p video from Vimeo over HTTPS. Video streaming generates more traffic than web browsing.

Simulated CPUs. We evaluate SUIT on three CPU models:

$\mathcal{A}$: Intel Core i9-9900K with a single frequency and voltage domain and different core counts.

$\mathcal{B}$: AMD Ryzen 7 7700X with per-core frequency domains.

$\mathcal{C}$: Intel Xeon Silver 4208 with per-core frequency and voltage domains.

The subscript, e.g., $\mathcal{A}_1$ defines the number of utilized CPU cores for which the result is relevant. $\mathcal{X}_\infty$ denotes methods where the number of CPU cores are irrelevant on the result.

6.3 Efficiency and Performance Impact

Table 6 shows the results of the instruction trace-based evaluation with the CPUs $\mathcal{A}$ to $\mathcal{C}$. The following sections go into detail about the CPUs and operating strategies. Figure 16 shows the results for performance and efficiency of all tested applications on CPU $\mathcal{C}$ running the fV operating strategy.

Different Undervolting Offsets. The measurements in Table 2 show that the efficiency approximately doubles when decreasing the voltage offset from −70 mV to −97 mV. This can be explained with the quadratic voltage dependency in the dynamic power consumption of a CMOS circuit. The approximate doubling also translates to a system with SUIT.

Dependency on Domain Count. Besides the DVFS curve change delay, the performance of SUIT depends on two factors: whether the CPU has a single or per-core DVFS domains; and, if there is only a single DVFS domain, the number of cores. Only the emulation method is core-count independent. The impact of the core count on the efficiency and performance in discussed in Section 6.4.

6.4 fV Operating Strategy

The fV strategy changes between three p-states E, C_f and C_V (see Figure 4) to execute every workload as efficiently and fast as possible. It is used on CPUs with the same number of frequency and voltage domains: $\mathcal{A}$ and $\mathcal{C}$. The following numbers are from CPU $\mathcal{C}$, the results for $\mathcal{A}_1$ are analogous. Figure 16 shows the detailed results for the performance and efficiency impact of a −70 mV and −97 mV undervolt.

We discuss the results for −97 mV. The average efficiency gain over all SPEC CPU2017 benchmarks is +11 %, the median is +13 % while running on the efficient DVFS curve 72.7 % of the time. Applications that execute faultable instruction sparingly stay primarily on the efficient curve. 557.xz sees the highest efficiency gain. 97.1 % of the time it is on the efficient DVFS curve, resulting in a +2.75 % performance and +16.9 % efficiency gain (not shown). Applications that execute faultable instructions frequently are primarily on the C_V p-state. 520.omnetpp is on the efficient curve 3.2 % of the time. The performance impact of −0.13 % is negligible while the power consumption is reduced by −0.60 % resulting in a +0.47 % efficiency gain (not shown). Applications in-between, switch between E, C_f and C_V repeatedly, resulting in a small performance loss but a large reduction in power consumption. 502.gcc experiences the worst impact on the performance with −2.89 %. The reduction in power consumption by −11.5 % results in an efficiency gain of +9.67 % (not shown). It is on the efficient curve 76.6 % of the time.

Core Count Influence on CPU $\mathcal{A}$. CPU $\mathcal{A}$ has a single DVFS domain, which means that every process on each core influences the others. This has a negative impact on performance and efficiency. While $\mathcal{A}_1$ sees a +12 % average efficiency gain, it is reduced to +5.8 % on $\mathcal{A}_4$ (4 cores utilized). We see the highest efficiency gain in 557.xz, which is +17.1 % vs +13.8 % (not shown). The relative difference is smaller due to the overall fewer faultable instructions. These cause fewer DVFS curve switches, influencing other cores less. For benchmarks that are primarily on the conservative curve, multiple cores have a small impact due to little curve switching. The difference for 520.omnetpp is +0.52 % on $\mathcal{A}_1$ to +0.17 % on $\mathcal{A}_4$ (not shown). Overall, SUIT with DVFS curve switching unfolds its full potential on CPUs with per-core frequency and/or voltage domains. Laptop CPUs often only have up to 4 cores that tend to be underutilized given typical office or web browsing usage. But even with 4 fully utilized cores SUIT has a small edge on the efficiency with +5.8 % on average.

Optimal Operating-Strategy Parameters. The operating strategy is configured with four parameters, see Section 4.3. We ran hundreds of simulations to find the optimal values to maximize the efficiency gain and used them for the evaluation. The result is shown in Table 7. We did not see a large impact of the parameters on the efficiency in a large range of values. Varying, e.g., the deadline ±10 µs changes the average efficiency by only -0.61 %. This indicates that workloads tolerate a range rather than requiring individual parameters, which simplifies SUIT as an operating system policy.

6.5 Frequency Operating Strategy on $\mathcal{B}$

CPU $\mathcal{B}$ has per-core frequency domains but a single voltage domain. To be independent of the number of cores, it uses the frequency to change the DVFS curve ($E \leftrightarrow C_f$). Changing the frequency takes 30 times longer than on CPU $\mathcal{A}$ and $\mathcal{C}$, explaining the large impact on performance and efficiency.

6.6 Instruction Emulation

In Table 6, operating strategy e shows the results for instruction emulation. For brevity we did not include emulation on CPU $\mathcal{C}$ as the results are very similar to CPU $\mathcal{A}$. On $\mathcal{A}$

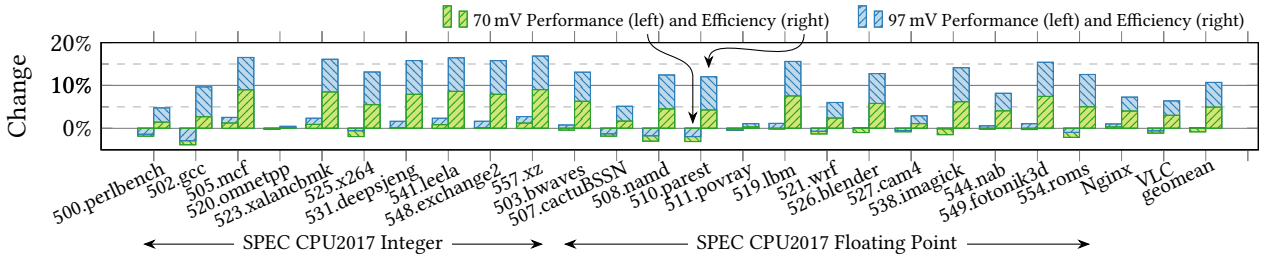

Figure 16. The performance and efficiency impact of SUIT on CPU $\mathcal{C}$ running the fV operating strategy.

Table 6. Power saving and performance impact of SUIT running on different CPUs with different operating strategies (OS) at a −70 mV and −96 mV undervolt. 525.x264 is shown explicitly as it is most adversely impacted by increasing the `IMUL` latency. All colored cells show an increase in efficiency or performance, where darker shades indicate the highest benefit in the respective column.

			70 mV Undervolt						97 mV Undervolt					
CPU	cores OS		$\text{SPEC}_{\text{gmean}}$	$\text{SPEC}_{\text{median}}$	525.x264	$\text{SPEC}_{\text{noSIMD}}$	Nginx	VLC	$\text{SPEC}_{\text{gmean}}$	$\text{SPEC}_{\text{median}}$	525.x264	$\text{SPEC}_{\text{noSIMD}}$	Nginx	VLC
$\mathcal{A}_1$	fV	Pwr	−5.6 %	−7.1 %	−7.1 %	−7.1 %	−3.5 %	−3.9 %	−9.7 %	−11 %	−12 %	−15 %	−5.8 %	−6.3 %
		Perf.	−0.2 %	−1.3 %	−1.3 %	+3.0 %	+0.5 %	−0.4 %	+0.8 %	+1.3 %	0.1 %	+3.4 %	+1.2 %	+0.2 %
		Eff.	+5.7 %	+6.2 %	+6.2 %	+11 %	+4.2 %	+3.6 %	+12 %	+14 %	+14 %	+21 %	+7.4 %	+6.9 %
$\mathcal{A}_4$	fV	Pwr	−4.6 %	−0.1 %	−6.9 %	−7.4 %	−1.0 %	−1.0 %	−8.9 %	−8.7 %	−13 %	−16 %	−1.6 %	−1.6 %
		Perf.	−3.9 %	−0.0 %	−7.9 %	+1.8 %	−0.3 %	−0.6 %	−3.6 %	−3.5 %	−7.2 %	+1.8 %	−0.1 %	−0.5 %
		Eff.	+0.7 %	0.1 %	−1.0 %	+10.0 %	+0.7 %	+0.4 %	+5.8 %	+5.7 %	+6.7 %	+22 %	+1.5 %	+1.1 %
$\mathcal{A}_\infty$	e	Pwr	−7.5 %	−7.6 %	−5.4 %	−7.5 %	−7.2 %	−7.2 %	−12 %	−12 %	−10 %	−17 %	−12 %	−12 %
		Perf.	−42 %	−12 %	+6.2 %	+1.4 %	−98 %	−92 %	−42 %	−12 %	+6.1 %	+1.4 %	−98 %	−92 %
		Eff.	−37 %	−4.5 %	+12 %	+9.6 %	−98 %	−91 %	−34 %	+0.6 %	+18 %	+22 %	−98 %	−91 %
$\mathcal{B}_\infty$	f	Pwr	−8.1 %	−7.8 %	−7.8 %	−9.1 %	−4.4 %	−4.4 %	−12 %	−11 %	−11 %	−14 %	−6.7 %	−6.7 %
		Perf.	−7.8 %	−7.8 %	−9.2 %	+0.4 %	−2.5 %	−2.5 %	−10 %	−11 %	−12 %	+0.6 %	−2.3 %	−2.3 %
		Eff.	+0.3 %	−0.0 %	−1.6 %	+11 %	+2.0 %	+2.0 %	+1.4 %	0.1 %	−1.6 %	+17 %	+4.7 %	+4.7 %
	e	Pwr	−9.2 %	−8.0 %	−11 %	−9.2 %	−9.8 %	−9.8 %	−14 %	−13 %	−16 %	−14 %	−15 %	−15 %
		Perf.	−26 %	−5.1 %	+15 %	−0.5 %	−96 %	−80 %	−26 %	−5.2 %	+19 %	0.0 %	−96 %	−80 %
		Eff.	−19 %	+3.1 %	+28 %	+9.5 %	−95 %	−78 %	−14 %	+9.3 %	+41 %	+17 %	−95 %	−76 %
$\mathcal{C}_\infty$	fV	Pwr	−5.6 %	−7.1 %	−7.1 %	−6.1 %	−3.6 %	−4.0 %	−9.8 %	−11 %	−12 %	−14 %	−5.8 %	−6.6 %
		Perf.	−0.8 %	−1.9 %	−1.9 %	+3.5 %	+0.3 %	−1.1 %	+0.2 %	+0.2 %	−0.6 %	+3.8 %	+1.0 %	−0.6 %
		Eff.	+5.1 %	+5.5 %	+5.5 %	+10 %	+4.0 %	+3.0 %	+11 %	+13 %	+13 %	+21 %	+7.3 %	+6.4 %

Table 7. The optimal parameters for the fV operating strategy and trashing prevention (see Section 4.3) used for all simulations.

CPU	p_dl	p_ts	p_ec	p_df
$\mathcal{A}$ & $\mathcal{C}$	30 µs	450 µs	3	14
$\mathcal{B}$	700 µs	14 ms	4	9

the efficiency impact is −34 % on average. A few dominant negative results skew the geometric mean so that the median efficiency increase over all benchmarks is +0.6 %. With fV, all SPEC benchmarks show increased efficiency, ranging from +0.52 % to +17.1 %, σ = 5.9. With emulation, efficiency ranges from −91.2 % to +27.0 %, σ = 36.6.

The performance of emulation depends only on instruction frequency but not distribution. This becomes evident when comparing the results with fV on CPU $\mathcal{A}$: Nginx experiences the largest difference in efficiency: +7.4 % with fV but −98 % with emulation. The short bursts of many AES encryptions are good for DVFS curve switching but impose prohibitive costs when every single encryption causes a exception for emulation.

525.x264 has the overall highest efficiency gain of +41 % on CPU $\mathcal{B}$. To simulate instruction emulation we include the benchmark results without SIMD instructions from Section 6.7, there 525.x264 is 22 % faster. This is combined with the 20 % efficiency increase at −97 mV (Section 5.4) and the short exception delay of CPU $\mathcal{B}$ (Section 5.3).

The threshold for a positive efficiency impact is approximately one disabled instructions per 4.1×10^{10} instructions. An exact threshold is impossible to define because the distribution of the instructions and the complexity of the emulated instructions have an impact as well. This shows that the viability of instruction emulation is highly dependent on the workload. However, due the hardware-software co-design of SUIT, the operating system can dynamically choose the best operating strategy for each workload.

Table 8. The number of SPEC CPU2017 benchmarks where compiling *without* SIMD instructions or using SIMD instructions and having the overhead of SUIT leads to higher performance at −97 mV. Emulation is always worse but does not require recompilation.

	$\mathcal{A}_1$	$\mathcal{A}_4$	$\mathcal{A}_\infty$	$\mathcal{B}_\infty$		$\mathcal{C}_\infty$
	fV	fV	e	f	e	fV
No SIMD	15	21	23	21	23	16
SUIT	8	2	0	2	0	7

6.7 SPEC without SIMD Instructions

In Section 5.8, we compared SPEC CPU2017 compiled with and without SIMD instructions. We now use these numbers to evaluate if compiling an application without SIMD is more efficient than using SUIT to execute them securely.

The benchmarks using SIMD have an overhead from SUIT while the others do not but observe the overhead from not using SIMD, see Section 5.8. We compare these results and then individually conduct a sensitivity study assessing the performance before computing the mean over all benchmarks. Table 6 shows the results in the column $\text{SPEC}_{\text{noSIMD}}$. The average score over SPEC CPU2017 on CPU $\mathcal{C}$ is +21 %. Emulation is always worse as it incurs the same overhead as $\text{SPEC}_{\text{noSIMD}}$ plus the emulation call overhead but does not require recompilation. Thus, in the emulation $\text{SPEC}_{\text{noSIMD}}$ column no instructions are emulated, i.e., the column shows the average if *all* benchmarks are compiled without SIMD.

Table 8 shows how often the performance benefits from the compile-time decision to disable SIMD instructions. On CPU $\mathcal{B}$ this is almost always the case due to its long frequency change delay. However, compiling without SIMD instructions is very unfavorable for some SPEC CPU2017 benchmarks. The worst case on CPU $\mathcal{C}$ is 508.namd. The efficiency increases by +12.4 % with SUIT but decreases by −20.5 % when compiling without SIMD instructions.

It is crucial to reiterate that these results do *not* translate to CPUs without SUIT. Only with SUIT is the security and correctness of the running applications guaranteed when undervolting. Without such guarantees, any comparison is deemed incorrect as it compromises security. Nonetheless, these results show that having less #DO exceptions and DVFS curve switches can be favorable for many benchmarks.

6.8 Performance and Efficiency Impact – Summary

On CPUs with one DVFS domain SUIT does not perform well if all cores are fully utilized. However, it still achieves a slight efficiency gain. On CPUs like this, emulation is better for some benchmarks, but it has a very high variance in respect to different workloads. The variance of the fV operating strategy is lower. It is a good "one fits all" approach because it stays on the C_V p-state for workloads with a high number of potentially faulting instructions, inducing negligible performance overheads. SUIT could dynamically switch between C_V and e for highest efficiency. The frequency change delay of CPU $\mathcal{B}$ is too high to work properly with SUIT. Emulation is more efficient than on $\mathcal{A}$ and $\mathcal{C}$ due to the shorter exception delay. A strategy that uses emulation for some workloads and deactivates SUIT for other workloads is the best compromise. For many benchmarks compiling them without SIMD instructions is most efficient. But this requires recompilation for CPUs with SUIT. On CPUs without SUIT this can be a disadvantage requiring different builds.

Due to the quadratic voltage dependency of the power consumption the undervolting offset from 20 % of the aging guardband has a large impact. But even without it SUIT overall increases the efficiency by 5 % to 10 %.

6.9 Security Analysis

The security of SUIT rests on the security of changing the CPU's DVFS curve and increasing the `IMUL` latency. An absolute argument or proof on the security of a specific approach to DVFS may be infeasible in the presence of process variation. It would also, security-wise, go beyond the guarantees vendors provide for current CPUs. Instead, we present reductionist arguments for each strategy showing that the security with SUIT is equivalent to the security of current CPUs.

SUIT. With SUIT, we disable specific instructions, so that they cannot fault. This removes these instructions from the optimization vendors perform to determine the optimal number of cycles per instruction and clock frequency, yielding a more efficient DVFS curve. For the remaining instructions, the security level is exactly the same as for current CPUs, as the vendor optimizes these instructions as usual and provides a DVFS curve for them. When a disabled instruction has to be executed, we resort to a voltage-frequency level the vendor determined to be stable in an optimization including these instructions. SUIT implements this by lowering the frequency (or increasing the voltage). Thus, in summary, the security of SUIT is based on the processes that are already established by vendors but now performed separately for conservative and efficient curves, i.e., without and with considering the set of faultable instructions.

Increased `IMUL` Latency. We only found the `IMUL` instructions to be too frequent to trap efficiently. Increasing the latency of an instruction increases the timing slack proportionally. Increasing the latency for `IMUL` from 3 to 4 leads to 33 % of the latency being slack time [49]. Increasing the clock frequency, on the other hand, reduces the instruction slack proportionally, e.g., increasing it by 25 % reduces the slack space for each instruction by 20 % [49]. We can also reduce the voltage accordingly but the relation with the voltage is not linear. Figure 13 shows the DVFS frequency-voltage pairs on an i9-9900K. We can see that, in the best case at 5 GHz, instead of a 33 % increase in frequency we can reduce the voltage by 220 mV. In the worst case, at very low frequencies, the voltage reduction is negligible as the power consumption of the CPU is not a concern. Reducing the voltage accordingly would lead to the same security as modern

CPUs currently have. Hence, security-wise, if we do not decrease the voltage further than indicated by these real-world numbers, we do not lower the security of the system.

7 Related Work

Numerous prior works investigated undervolting [6, 7, 16, 30, 35, 36, 43, 51, 52]. What differentiates them from SUIT is that they undervolt by removing or shrinking the CPU's guardband. In most of them, the impact of aging is out of scope. Because SUIT shrinks the voltage from the variation in the voltage requirements of instructions, the CPU keeps the voltage guardband required for aging fully or most of it.

xDVS. Koutsovasilis et al. [36] observed that the minimum CPU voltage depends on the workload. Their extended dynamic voltage scaling (xDVS) governor scales the voltage according to the workload by taking performance monitor counter readings into account. They observe over 40 % energy savings by undervolting the CPU by over 200 mV.

CADU++. Maroudas et al. [43] additionally differentiate between kernel and user space for their workload dependent undervolting, CADU++. They observed that the undervolting potential of user space code is consistently higher than that of kernel code. On average they achieve over 240 mV undervolting resulting in up to 30 % power savings. However, CADU++ requires a voltage change on every kernel entry and exit. According to our measurements, they ignore parts of the voltage change delay by only considering the delay to write the MSR, and not until the voltage actually changes.

ECC Cache Errors. Bacha et al. [7] base their work on cache lines faulting first when undervolting Itanium CPUs. We did not observe this on x86. The faults are corrected with ECCs. A calibration phase finds the weakest cache line and the faulting voltage level, which can be repeated periodically to counter aging. They achieved a 33 % power reduction.

Razor. Ernst et al. [13] proposed Razor. It follows the elegant idea of using slightly skewed clock edges and shadow circuitry to detect when critical paths in circuits are about to violate their timing constraints. This allows Razor to dynamically find the optimal voltage and frequency for every chip. While Razor promises substantial energy savings, the special Razor circuits increase the complexity significantly. Razor has still found no adoption in commercial systems today. In comparison, SUIT comprises a rather simple set of changes without any complexity increase on the circuit level.

Efficient Scheduling. Lawall et al. [38] group cores in a *nest* to keep their clock frequencies high and schedule tasks that share resources in proximity. This minimizes clock frequency changes and sharing of data between sockets. 2- and 4-socket multicore machines gain up to 20 % performance and efficiency. Gouicem et al. [18] design a scheduler that minimizes frequency changes. They gain up to 56 % performance on an 80-core Intel Xeon CPU. Similar scheduling methods could also be used in conjunction with SUIT to minimize DVFS curve changes.

Heterogeneous CPUs house sets of cores of different types and capabilities. Examples are recent Intel CPUs with power and efficiency cores [48] and ARMs big.LITTLE [5] design. With large differences in power consumption between power and efficiency cores, by design, they lack support for dynamic adjustment of the number of cores for each type. SUIT dynamically adapts to workloads by running any number of cores with the conservative or efficient DVFS curves.

8 Discussion

Protection against Undervolting Attacks. This work is primarily aimed at providing a benign OS with a method to securely reduce the power consumption of a computer. We did not investigate if SUIT can also protect against undervolting attacks [31, 47, 56, 60] by only allowing DVFS curve switching if all faultable instructions are disabled.

Speculative Execution. Speculative out-of-order execution of disabled instructions must not happen. This could cause exploits like meltdown where the exception is only thrown at retirement, which is too late. The 4 clock cycle IMUL is not a faultable instruction and can be speculatively executed.

Side-Channel Leakage. An attacker could learn when disabled instructions are executed to build a covert channel [22].

9 Conclusion

In this paper we proposed SUIT, a technique enabling substantial efficiency gains on modern CPUs. It is based on the fact that only a small subset of instructions requires the conservative power margins observed on current CPUs. With SUIT we disable the execution of faultable instructions and run the CPU on a more efficient DVFS curve 72.7 % of the time, increasing the efficiency by 11.0 % with no performance impact over SPEC CPU2017, without reducing the system's reliability or security. Together with compile-time optimizations for SUIT the CPU efficiency increases by 20.8 % while the performance increases by 3.79 %. Hence, we conclude that SUIT is a practical solution to increase energy efficiency.

Acknowledgements

We thank Andreas Kogler, Sarah Rinderer, the reviewers and especially our shepherd, Guilherme Cox, for their feedback and help. This research is supported in part by the European Research Council (ERC project FSSec 101076409), the Austrian Science Fund (FWF SFB project SPyCoDe F8504), NSF grants 1813004, 2217020, 2316201 and a research grant from CISCO. This work has benefitted from Dagstuhl Seminar 22341 (PEACHES). Additional funding was provided by generous gifts from Red Hat, Google and Intel. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the funding parties.

References

[1] AMD. Processor Programming Reference (PPR) for AMD Family 17h Model 01h, Revision B1 Processors, 2017.

[2] AMD. *Software Optimization Guide for AMD Family 17h Processors*, 6 2017.

[3] AMD. Software Optimization Guide for AMD EPYC 7003 Processors, 11 2020.

[4] AMD. Processor Programming Reference (PPR) for AMD Family 19h Model 21h, Revision B0 Processors, 2021.

[5] ARM. big.LITTLE Technology: The Future of Mobile, 2013. URL: https://armkeil.blob.core.windows.net/developer/Files/pdf/white-paper/big-little-technology-the-future-of-mobile.pdf.

[6] Anys Bacha and Radu Teodorescu. Dynamic reduction of voltage margins by leveraging on-chip ecc in itanium ii processors. In *International Symposium on Computer Architecture (ISCA)*, 2013. doi:10.1145/2485922.2485948.

[7] Anys Bacha and Radu Teodorescu. Using ecc feedback to guide voltage speculation in low-voltage processors. In *International Symposium on Microarchitecture (MICRO)*, 2014. doi:10.1109/MICRO.2014.54.

[8] Alessandro Barenghi, Guido Bertoni, Emanuele Parrinello, and Gerardo Pelosi. Low Voltage Fault Attacks on the RSA Cryptosystem. In *Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, 2009.

[9] Fabrice Bellard. QEMU, a Fast and Portable Dynamic Translator. In *USENIX ATC*, 2005.

[10] James Bucek, Klaus-Dieter Lange, and Jóakim v. Kistowski. SPEC CPU2017: Next-Generation Compute Benchmark. In *International Conference on Performance Engineering*, 2018. doi:10.1145/3185768.3185771.

[11] Ian Cutress. AMD Precision Boost Overdrive 2: Adaptive Undervolting For Ryzen 5000 Coming Soon, 11 2020. URL: https://www.anandtech.com/show/16267/amd-precision-boost-overdrive-2-adaptive-undervolting-for-ryzen-5000-coming-soon.

[12] Dell. Intel i9 Processor Throttling Under AVX (Advanced Vector eXtensions), 2021. URL: https://www.dell.com/support/kbdoc/en-us/000184687/intel-i9-processor-throttling-under-avx-advanced-vector-extensions.

[13] Dan Ernst, Nam Sung Kim, Shidhartha Das, Sanjay Pant, Rajeev Rao, Toan Pham, Conrad Ziesler, David Blaauw, Todd Austin, Krisztian Flautner, et al. Razor: A Low-Power Pipeline Based on Circuit-Level Timing Speculation. In *MICRO*, 2003. doi:10.1145/1150343.1150348.

[14] Nicholas Fearn. Will AWS pledge to extend life of servers inspire other cloud firms to follow suit?, 5 2022. URL: https://www.computerweekly.com/feature/Will-AWS-pledge-to-extend-life-of-servers-inspire-other-cloud-firms-to-follow-suit.

[15] Agner Fog. The microarchitecture of Intel, AMD, and VIA CPUs: An optimization guide for assembly programmers and compiler makers, 2021. URL: https://www.agner.org/optimize/microarchitecture.pdf.

[16] Dimitris Gizopoulos, George Papadimitriou, Athanasios Chatzidimitriou, Vijay Janapa Reddi, Behzad Salami, Osman S Unsal, Adrian Cristal Kestelman, and Jingwen Leng. Modern hardware margins: Cpus, gpus, fpgas recent system-level studies. In *International Symposium on On-Line Testing and Robust System Design (IOLTS)*, 2019. doi:10.1109/IOLTS.2019.8854386.

[17] Corey Gough, Ian Steiner, Winston Saunders, Corey Gough, Ian Steiner, and Winston Saunders. CPU Power Management. *Energy Efficient Servers: Blueprints for Data Center Optimization*, 2015. doi:10.1007/978-1-4302-6638-9_2.

[18] Redha Gouicem, Damien Carver, Jean-Pierre Lozi, Julien Sopena, Baptiste Lepers, Willy Zwaenepoel, Nicolas Palix, Julia Lawall, and Gilles Muller. Fewer Cores, More Hertz: Leveraging High-Frequency Cores in the OS Scheduler for Improved Application Performance. In *USENIX Annual Technical Conference (USENIX ATC)*, 2020.

[19] Xinfei Guo, Vaibhav Verma, Patricia Gonzalez-Guerrero, and Mircea R Stan. When "things" Get Older: Exploring Circuit Aging in IoT Applications. In *International Symposium on Quality Electronic Design (ISQED)*, 2018. doi:10.1109/ISQED.2018.8357304.

[20] Daniel Hackenberg, Thomas Ilsche, Robert Schöne, Daniel Molka, Maik Schmidt, and Wolfgang E. Nagel. Power measurement techniques on standard compute nodes: A quantitative comparison. In *IEEE ISPASS*, 2013. doi:10.1109/ISPASS.2013.6557170.

[21] Daniel Hackenberg, Robert Schöne, Thomas Ilsche, Daniel Molka, Joseph Schuchart, and Robin Geyer. An energy efficiency feature survey of the intel haswell processor. In *International Parallel and Distributed Processing Symposium Workshop (IPDPSW)*, 2015. doi:10.1109/IPDPSW.2015.70.

[22] Jawad Haj-Yahya, Lois Orosa, Jeremie S Kim, Juan Gómez Luna, A Giray Yağlıkçı, Mohammed Alser, Ivan Puddu, and Onur Mutlu. Ichannels: Exploiting current management mechanisms to create covert channels in modern processors. In *ISCA*, 2021. doi:10.1109/ISCA52012.2021.00081.

[23] Sebastian Herbert and Diana Marculescu. Analysis of dynamic voltage/frequency scaling in chip-multiprocessors. In *International Symposium on Low Power Electronics and Design (ISLPED)*, 2007. doi:10.1145/1283780.1283790.

[24] Vincent Huard, Souhir Mhira, A Barclais, X Lecocq, F Raugi, M Cantournet, and Alain Bravaix. Managing electrical reliability in consumer systems for improved energy efficiency. In *IEEE International Reliability Physics Symposium (IRPS)*, 2018. doi:10.1109/IRPS.2018.8353561.

[25] Igor Sysoev. Advanced Load Balancer, Web Server, & Reverse Proxy - NGINX, 2004. URL: https://www.nginx.com/.

[26] Intel. Intel 64 and IA-32 Architectures Software Developer's Manual, Volume 1: Basic Architecture, 2016.

[27] Intel. Intel 64 and IA-32 Architectures Software Developer's Manual, Volume 3 (3A, 3B & 3C): System Programming Guide, 2019.

[28] Intel. Changes in Customer Support and Servicing Updates for Select Intel® Processors, 2023. URL: https://www.intel.com/content/www/us/en/support/articles/000022396/processors.html.

[29] Nicola Jones. How to stop data centres from gobbling up the world's electricity. *Nature*, 2018. doi:10.1038/d41586-018-06610-y.

[30] Manolis Kaliorakis, Athanasios Chatzidimitriou, George Papadimitriou, and Dimitris Gizopoulos. Statistical analysis of multicore cpus operation in scaled voltage conditions. *IEEE Computer Architecture Letters*, 2018. doi:10.1109/LCA.2018.2798604.

[31] Zijo Kenjar, Tommaso Frassetto, David Gens, Michael Franz, and Ahmad-Reza Sadeghi. V0LTpwn: Attacking x86 Processor Integrity from Software. In *USENIX Security*, 2020.

[32] Andreas Kogler, Daniel Gruss, and Michael Schwarz. Minefield: A Software-only Protection for SGX Enclaves against DVFS Attacks. In *USENIX Security*, 2022.

[33] Andreas Kogler, Jonas Juffinger, Lukas Giner, Lukas Gerlach, Martin Schwarzl, Michael Schwarz, Daniel Gruss, and Stefan Mangard. Collide+power: Leaking inaccessible data with software-based power side channels. In *USENIX Security*, 2023.

[34] Martijn Koot and Fons Wijnhoven. Usage impact on data center electricity needs: A system dynamic forecasting model. *Applied Energy*, 2021. doi:10.1016/j.apenergy.2021.116798.

[35] Panos Koutsovasilis, Christos D Antonopoulos, Nikolaos Bellas, Spyros Lalis, George Papadimitriou, Athanasios Chatzidimitriou, and Dimitris Gizopoulos. The Impact of CPU Voltage Margins on Power-Constrained Execution. *IEEE Transactions on Sustainable Computing*, 2020. doi:10.1109/TSUSC.2020.3045195.

[36] Panos Koutsovasilis, Konstantinos Parasyris, Christos D Antonopoulos, Nikolaos Bellas, and Spyros Lalis. Dynamic undervolting to improve energy efficiency on multicore x86 cpus. *IEEE Transactions on Parallel and Distributed Systems*, 2020. doi:10.1109/TPDS.2020.3004383.

[37] Vlad Krasnov. On the dangers of Intel's frequency scaling), 2017. URL: https://blog.cloudflare.com/on-the-dangers-of-intels-frequency-scaling/.

[38] Julia Lawall, Himadri Chhaya-Shailesh, Jean-Pierre Lozi, Baptiste Lepers, Willy Zwaenepoel, and Gilles Muller. OS Scheduling with Nest: Keeping Tasks Close Together on Warm Cores. In *Proceedings of the Seventeenth European Conference on Computer Systems*, 2022. doi:10.1145/3492321.3519585.

[39] Moritz Lipp, Andreas Kogler, David Oswald, Michael Schwarz, Catherine Easdon, Claudio Canella, and Daniel Gruss. PLATYPUS: Software-based Power Side-Channel Attacks on x86. In *S&P*, 2021. doi:10.1109/SP40001.2021.00063.

[40] S Mahapatra, N Goel, S Desai, S Gupta, B Jose, S Mukhopadhyay, K Joshi, A Jain, AE Islam, and MA Alam. A Comparative Study of Different Physics-Based NBTI Models. *IEEE transactions on electron devices*, 2013. doi:10.1109/TED.2013.2238237.

[41] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. Springer Science & Business Media, 2008.

[42] Aniruddha Marathe, Yijia Zhang, Grayson Blanks, Nirmal Kumbhare, Ghaleb Abdulla, and Barry Rountree. An empirical survey of performance and energy efficiency variation on intel processors. In *International Workshop on Energy Efficient Supercomputing*, 2017. doi:10.1145/3149412.3149421.

[43] Emmanouil Maroudas, Spyros Lalis, Nikolaos Bellas, and Christos D Antonopoulos. Exploring the potential of context-aware dynamic cpu undervolting. In *ACM International Conference on Computing Frontiers*, 2021. doi:10.1145/3457388.3458658.

[44] Ken Martin. *Digital integrated circuit design*. 2000.

[45] Eleršič Miha. Guide to linux undervolting for Haswell and never Intel CPUs, 2018. URL: https://github.com/mihic/linux-intel-undervolt.

[46] Evelyn Mintarno, Joëlle Skaf, Rui Zheng, Jyothi Bhaskar Velamala, Yu Cao, Stephen Boyd, Robert W Dutton, and Subhasish Mitra. Self-Tuning for Maximized Lifetime Energy-Efficiency in the Presence of Circuit Aging. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2011. doi:10.1109/TCAD.2010.2100531.

[47] Kit Murdock, David Oswald, Flavio D. Garcia, Jo Van Bulck, Daniel Gruss, and Frank Piessens. Plundervolt: Software-based Fault Injection Attacks against Intel SGX. In *S&P*, 2020. doi:10.1109/SP40000.2020.00057.

[48] Rukmabhatla Nikhil, Chabukswar Rajshree, Gohad Sneha, and Chynoweth Michael. Introduction to Performance Hybrid Architecture for 12th Generation Intel Core Processors, 2013. URL: https://www.intel.com/content/www/us/en/content-details/685861/introduction-to-performance-hybrid-architecture-for-12th-generation-intel-core-processors.html.

[49] Vojin G Oklobdzija, Vladimir M Stojanovic, Dejan M Markovic, and Nikola M Nedovic. *Digital System Clocking: High-Performance and Low-Power Aspects*. Wiley-IEEE Press, 2005.

[50] George Papadimitriou, Athanasios Chatzidimitriou, and Dimitris Gizopoulos. Adaptive voltage/frequency scaling and core allocation for balanced energy and performance on multicore cpus. In *IEEE international symposium on high performance computer architecture (HPCA)*. IEEE, 2019. doi:10.1109/HPCA.2019.00033.

[51] George Papadimitriou, Manolis Kaliorakis, Athanasios Chatzidimitriou, Dimitris Gizopoulos, Peter Lawthers, and Shidhartha Das. Harnessing voltage margins for energy efficiency in multicore cpus. In *IEEE/ACM International Symposium on Microarchitecture*, 2017. doi:10.1145/3123939.3124537.

[52] George Papadimitriou, Manolis Kaliorakis, Athanasios Chatzidimitriou, Charalampos Magdalinos, and Dimitris Gizopoulos. Voltage margins identification on commercial x86-64 multicore microprocessors. In *IEEE Symposium on On-Line Testing (IOLTS)*, 2017. doi:10.1109/IOLTS.2017.8046198.

[53] Devyani Patra, Jiayang Zhang, Runsheng Wang, Mehdi Katoozi, Ethan H Cannon, Ru Huang, and Yu Cao. Compact modeling and simulation of accelerated circuit aging. In *IEEE Custom Integrated Circuits Conference (CICC)*, 2018. doi:10.1109/CICC.2018.8357063.

[54] Michael K Patterson. The Effect of Data Center Temperature on Energy Efficiency. In *Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems*, 2008. doi:10.1109/ITHERM.2008.4544393.

[55] Pablo Prieto, Pablo Abad, Jose Angel Herrero, Jose Angel Gregorio, and Valentin Puente. SPECcast: A Methodology for Fast Performance Evaluation with SPEC CPU 2017 Multiprogrammed Workloads. In *ICPP*, 2020. doi:10.1145/3404397.3404424.

[56] Pengfei Qiu, Dongsheng Wang, Yongqiang Lyu, and Gang Qu. VoltJockey: Breaching TrustZone by Software-Controlled Voltage Manipulation over Multi-core Frequencies. In *CCS*, 2019. doi:10.1145/3319535.3354201.

[57] Josyula R Rao and Pankaj Rohatgi. EMpowering Side-Channel Attacks. *Cryptology ePrint Archive, Report 2006/037*, 2001.

[58] Christian Schlünder, Stefano Aresu, Georg Georgakos, Werner Kanert, Hans Reisinger, Karl Hofmann, and Wolfgang Gustin. HCI vs. BTI?-Neither one's out. In *IEEE International Reliability Physics Symposium (IRPS)*, 2012. doi:10.1109/IRPS.2012.6241797.

[59] D Suleiman, M Ibrahim, and I Hamarash. Dynamic voltage frequency scaling (dvfs) for microprocessors power and energy reduction. In *International Conference on Electrical and Electronics Engineering*, 2005.

[60] Adrian Tang, Simha Sethumadhavan, and Salvatore Stolfo. CLKSCREW: Exposing the Perils of Security-Oblivious Energy Management. In *USENIX Security*, 2017.

[61] Victor M Van Santen, Hussam Amrouch, Narendra Parihar, Souvik Mahapatra, and Jörg Henkel. Aging-Aware Voltage Scaling. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2016.

[62] VideoLan. VLC media player, 2006. URL: https://www.videolan.org/vlc/index.html.

[63] Yingchen Wang, Riccardo Paccagnella, Elizabeth He, Hovav Shacham, Christopher W. Fletcher, and David Kohlbrenner. Hertzbleed: Turning Power Side-Channel Attacks Into Remote Timing Attacks on x86. In *USENIX Security*, 2022.

[64] Will Glozer. wrk - a HTTP benchmarking tool, 2015. URL: https://github.com/wg/wrk.

[65] MF Zainudin, H Hussin, AK Halim, and J Karim. Aging analysis of high performance FinFET flip-flop under Dynamic NBTI simulation configuration. In *IOP Conference Series: Materials Science and Engineering*, 2018. doi:10.1088/1757-899X/341/1/012012.

[66] Zuodong Zhang, Zizheng Guo, Yibo Lin, Meng Li, Runsheng Wang, and Ru Huang. AVATAR: An Aging-and Variation-Aware Dynamic Timing Analyzer for Error-Efficient Computing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2023. doi:10.1109/TCAD.2023.3255167.

[67] Zuodong Zhang, Runsheng Wang, Xuguang Shen, Dehuang Wu, Jiayang Zhang, Zhe Zhang, Joddy Wang, and Ru Huang. Aging-Aware Gate-Level Modeling for Circuit Reliability Analysis. *IEEE Transactions on Electron Devices*, 2021. doi:10.1109/TED.2021.3096171.

A Artifact Appendix

A.1 Abstract

We provide artifacts to measure key results to perform the trace-based simulation resulting in the numbers in Table 6. While we provide artifacts to reproduce all results as well as all our measurement and simulation results, only a subset of the experiments is described in this document and part of the artifact reproducibility evaluation.

A.2 Artifact check-list (meta-information)

- **Program: SPEC-CPU2017 (proprietary, not included)**
- **Compilation: gcc / g++ version ≥ 9**
- **Run-time environment: debian-based Linux, root access**
- **Hardware: Intel CPU up until 9th generation**
- **Execution: Sole user, process pinning**
- **Metrics: Power savings, performance impact**
- **Experiments: CPU Microbenchmarks and Simulation**
- **How much disk space required (approximately)?:** ≈ 200 GB
- **How much time is needed to prepare workflow (approximately)?:** 4 h
- **How much time is needed to complete experiments (approximately)?:** 24 h
- **Publicly available?: Yes**
- **Code licenses (if publicly available)?: GPLv3 License**
- **Data licenses (if publicly available)?: MIT License**
- **Archived?: 10.5281/zenodo.10479442**

A.3 Description

A.3.1 How to access All code and data is available at 10.5281/zenodo.10479442. Due to license restrictions, the user has to provide an iso image of the SPEC CPU2017 benchmarks

A.3.2 Hardware dependencies An Intel CPU where undervolting with the 0x150 MSR still works (≤ 9th gen if SGX is disabled in the BIOS).

A.3.3 Software dependencies Linux distribution with the possibility to install kernel modules, we used Ubuntu 20.04. SPEC CPU 2017 iso image.

A.3.4 Data sets We provide some data that takes too long to record for the artifact evaluationluation. EHowever, verythingall code and instructions to record this data isare available.

A.4 Installation

Every experiment directory contains a Makefile. There are some build dependencies: `coreutils, build-essential, libarchive-dev, make,linux-tools-generic`

A.5 Experiment workflow

We provide artifacts to verify the following claims. Chained together they result in similar numbers to the numbers of Table 6. Due to process variations every CPU is affected differently by undervolting, therefore, the results will not exactly, i.e., they depend on the specific variations.

C1 Voltage Change Delay
C2 Frequency Change Delay
C3 Efficiency and Performance Impact of Undervolting
C4 Final Simulation

The gem5 simulations of the increased IMUL latency and the recording of the instruction traces of the SPEC CPU2017 benchmarks take hundreds of hours to run. We publish the results but also all artifacts to run these experiments.

All experiments come with a `README.md` file containing additional information that does not fit this document.

A.6 Disclaimer

CPU undervolting can cause instabilities! This can break file systems and cause data loss or corruption. Never run this software on a system without backups. We are not responsible for any damage caused by this software.

A.7 Evaluation and expected results

For the first three claims **C1** to **C3** we do not expect specific results. The simulation results of **C4** show that SUIT has an overall positive impact on CPU energy efficiency.

A.7.1 Voltage Change Delay

Measure the voltage change delay from Section 5.2.

Path: `5_microbenchmarks/1_voltage_change_delay`

Run: `./run.sh`

The run script first builds everything and then loads the kernel module. If successful, the measurement is run, measuring 20 voltage offset changes of −70 mV. Finally, it calls the plot script with the `result.csv` file to plot the data.

Plot: `python3 user/plot.py result.csv`

The plot script also prints the average time it takes to change the voltage to stdout. This time is required later to define the CPU for the simulation.

Result: Time it takes to change the core voltage.

A.7.2 Frequency Change Delay

Measure the voltage change delay from Section 5.2.

Path: `5_microbenchmarks/3_freq_change_delay`

Prepare: Turn off hardware controlled p-states (HWP) by booting the kernel with the `intel_pstate=passive` command line option.

Run: `./run.sh`

The script measures 20 frequency changes of −500 MHz from 3 GHz. Finally it calls the plot script with the `result.csv` file to plot the data.

Plot: `python3 user/plot.py result.csv`

The plot script also prints the average time it takes to change

the frequency to stdout. This time is required later to define the CPU for the simulation.

Result: Time it takes to change the frequency.

A.7.3 Efficiency and Performance Impact of Undervolting

Measure the efficiency and performance impact of undervolting from Section 5.4.

Path: `5.4_power_efficiency_and_performance`
The `README.md` in this directory contains additional details.

Prepare: There are multiple preparation steps:
Install SPEC CPU2017
Do not forget to `runcpu -update` to update the installation to the newest version.
Update the `SPEC_PATH` in the first line of `5.4_power_efficiency_and_performance/Makefile` to point to the SPEC CPU2017 installation directory.
Build with `make` and copy the built files with `make copy`. The later command requires root privileges to set the root sticky bit for the `measure` binary, because it must run as root but we do not want to start SPEC CPU2017 as root.
Test the SPEC CPU2017 installation with:

```
runcpu -config=power -size=test -define \
undervolting=0 specrate
```

Run: SPEC CPU2017 is run twice, once with the CPU at the default voltage and once with the CPU undervolted. The undervolt offset is defined in the `-define undervolting=X` command line argument when starting the benchmark.
Run with the default voltage:

```
runcpu -config=power -size=ref -n 2 \
-output_format config,csv -copies=$(nproc) \
-define undervolting=0 specrate
```

Run with a −70 mV offset:

```
runcpu -config=power -size=ref -n 2 \
-output_format config,csv -copies=$(nproc) \
-define undervolting=-70 specrate
```

Gather Results: The power measurement results are in `$SPEC_PATH/power_measurements` the benchmark scores in `$SPEC_PATH/results`. Combine them with:

```
mkdir -p results/spec
cp $SPEC_PATH/power_measurements/* results/
cp $SPEC_PATH/results/* results/spec
```

The files for the three CPUs from Table 2 are available at 10.5281/zenodo.10479442.

Plot: `./plot.py result`
This plots the changes in CPU frequency, voltage, power and benchmark scores. It prints the exact changes in % to stdout.

Result: In the last printed table, the relevant columns are: `pct` showing the benchmark score increase and `eff` showing the efficiency increase at −70 mV. We expect the benchmark scores and efficiency to increase.

A.7.4 Final Simulation

Perform the final simulation from Section 6.

Path: `6.2_instruction_trace_based_evaluation`

Prepare:
CPU Configuration File
Copy `cpus/i9-9900K-70mV.csv` to `cpus/cpu_name-70mV.csv` as a starting point. The file is then configured with the results from the previous experiments. An exact description of all fields is in the `README.md` file.

Instruction Traces
We provide the instruction traces for all benchmarks at 10.5281/zenodo.10479442 `instruction_traces.tar.gz` except for 520.omnetpp and 521.wrf, as they are too large.

Run: `./simulate_all.py cpu_name-70mV \ 30,15,3,14 voltfreq "" path/to/spec/traces/5*`

Postprocess: `./postprocess.py results_XXX.json`
The simulation creates an output `.json` file containing the results of each benchmark. Running the post-process script prints the results formatted as a table to stdout.

Result: The last three columns are the most relevant, the row `geomean` contains the numbers of Table 6:

- `power perc`: change in power consumption in %: "Pwr"
- `perf perc`: change in performance in %: "Perf."
- `eff perc`: change in efficiency in %: "Eff."

We expect the efficiency change to be positive to show that SUIT is viable. Additionally, all three numbers should be similar to numbers of $\mathcal{A}_1$ with the fV operating strategy and column $\text{SPEC}_{\text{gmean}}$. The performance impact is allowed to be slightly negative.

Because we cannot provide the traces for 520.omnetpp and 521.wrf the resulting mean is slightly better than with the two benchmarks included. On our tested CPU $\mathcal{A}$ at −70 mV, excluding the two benchmarks increases the efficiency gain from 5.7 % (see Table 6) to 6.1 %.

A.7.5 Regenerate Table 6

We are now able to generate Table 6 from the paper with the additional results from this evaluation.

Path: `6.2_instruction_trace_based_[...]/scripts`

Prepare: Update the file path at `generate_results_table.py:246` to point to the .json result file from the simulation.

Run: `./generate_results_table.py -full`

Result: `results_table.tex`
When compiled it shows Table 6 with an additional CPU $\mathcal{D}$ showing the results of this evaluation.