Energy Efficiency Aspects of the AMD Zen 2 Architecture

Robert Schöne[1] Thomas Ilsche[2] Mario Bielert[2] Markus Velten[2] Markus Schmidl[3] Daniel Hackenberg[2]
Center for Information Services and High Performance Computing (ZIH), TU Dresden, Dresden, 01062, Germany
[1] robert.schoene@tu-dresden.de [2] firstname.lastname@tu-dresden.de [3] markus.schmidl@mailbox.tu-dresden.de

***Abstract*—In High Performance Computing, systems are evaluated based on their computational throughput. However, performance in contemporary server processors is primarily limited by power and thermal constraints. Ensuring operation within a given power envelope requires a wide range of sophisticated control mechanisms. While some of these are handled transparently by hardware control loops, others are controlled by the operating system. A lack of publicly disclosed implementation details further complicates this topic. However, understanding these mechanisms is a prerequisite for any effort to exploit the full computing capability and to minimize the energy consumption of today's server systems. This paper highlights the various energy efficiency aspects of the AMD Zen 2 microarchitecture to facilitate system understanding and optimization. Key findings include qualitative and quantitative descriptions regarding core frequency transition delays, workload-based frequency limitations, effects of I/O die P-states on memory performance as well as discussion on the built-in power monitoring capabilities and its limitations. Moreover, we present specifics and caveats of idle states, wakeup times as well as the impact of idling and inactive hardware threads and cores on the performance of active resources such as other cores.**

***Index Terms*—AMD; Zen 2; Epyc Rome; power saving; energy efficiency; DVFS; C-State; performance; RAPL**

I. Introduction

With the Epyc Rome processor generation, AMD processors gained a noticeable share in the TOP500 list of supercomputers for the first time since Opteron Interlagos, which debuted in 2011. The new architecture is not only competitive in terms of performance, but also power efficiency among systems using general-purpose x86 processors as Figure 1 shows.

The Green500 [1] list, used to create Figure 1, ranks top High Performance Computing (HPC) systems by their energy efficiency under full load. In practice, *energy efficiency* is not only defined by power efficiency during peak performance. Many mechanisms, such as P-states or C-states, concern operation during scaled-down performance or idle phases. Other mechanisms, such as Turbo frequencies and power capping, aim at maximizing performance under thermal and power constraints. They are supported by internal energy measurements, which can also be used for energy-efficiency optimizations. This paper analyzes these dynamic and highly configurable mechanisms rather than the application-specific *performance per watt*. The resulting insight is the foundation to improve the complex interactions between applications, operating systems (OSs), and independent hardware control for performance and energy efficiency.

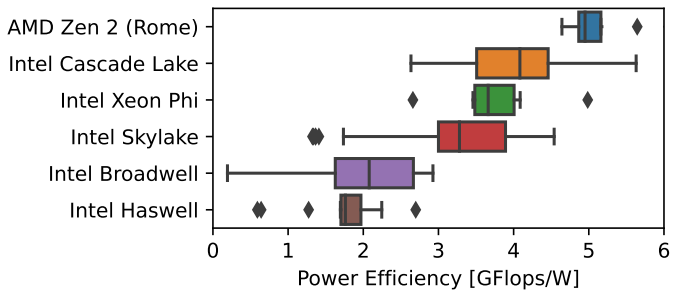


Fig. 1: Efficiency of systems with x86 processors in the 2021/07 Green500 list [1] (architectures with > 5 systems).

The paper is structured as follows: Section II and Section III discuss existing work on the evaluation of energy efficiency mechanisms and the Rome architecture, respectively. Section IV introduces our test system and power measurement infrastructure. The next three sections highlight particular aspects, each including methodologies, test results, and a discussion: In Section V, we unveil details on processor frequencies. Section VI covers characterizations of processor idle states. In Section VII, we validate the accuracy of the internal power monitoring mechanism. We conclude the paper with a summary and outlook in Section VIII.

II. Related Work on Efficiency Mechanisms

A. ACPI States

Power saving interfaces are defined in the Advanced Configuration and Power Interface (ACPI) [3]. Performance states (P-states) provide different performance levels and can be selected during runtime [3, Section 2.6]. Usually processors implement these with Dynamic Voltage and Frequency Scaling (DVFS). However, their particular implementation is highly processor-architecture dependent. Mazouz et al. were one of the first to investigate this in [4]: They describe how P-state transition times depend on initial and target processor frequency. We show in [5] that waking an idling processor core is also frequency-dependent, but additionally depends on the waker-wakee-relation and the applied idle state. Likewise, we also show in [6] how long it takes to enter an idle state. Finally, we describe the effect of clock modulation (throttling) in [7]. This paper covers P-state and C-state transitions in Section V and Section VI, respectively. Software controlled clock modulation, however, is not publicly documented for the Zen 2 architecture [2], [8].

B. Processor-internal Power Measurement and Capping

Processor-internal power monitoring is used to select turbo frequencies and implement power capping [9], [10]. The accuracy of these monitoring mechanisms therefore directly influences processor performance. We describe the Intel Running Average Power Limit (RAPL) for Intel Sandy Bridge and the AMD Application Power Management (APM) for AMD Bulldozer in [11]. We find that both are based on models that use data from processor internal resource usage monitors. We also analyze RAPL for Intel Haswell processors [12] and describe it to be accurate and based on measurements. Hähnel et al. measure the update rate of RAPL in [13] as 1 ms. Lipp et al. show in [14] that RAPL can provide significantly higher rates for the core power domain (pp0) for certain processors. They leverage this for a side channel attack [14].

C. Processor-specific Overviews

We describe the Intel Haswell server architecture in detail [12]. In addition to the previously mentioned RAPL analysis, the authors found an asynchronous mechanism that sets core frequencies in an interval of 500 µs. We also describe the interaction between core and uncore frequency mechanisms and show how concurrency and frequencies influence memory bandwidths. Gough et al. provide a broad overview of the Haswell architecture as well as suggestions for tuning server systems according to user requirements [15]. We describe the Intel Skylake server architecture [16], covering the internal hardware control for uncore frequencies, AVX-frequency mechanisms, and the influence of data on the power consumption of a well-defined workload. We show that uncore frequency changes can occur every 1.5 ms.

III. Architectural Details of "Rome" Processors

A. General Architecture Details

Zen 2 uses a modular design on multiple levels [2, Section 1.8.1]. The structure of the processor is depicted in Figure 2. Four cores are clustered in one Core Complex (CCX, also CPU Complex). One Core Complex Die (CCD) comprises two CCXs. Up to eight CCDs are attached to one I/O die on processors with up to 64 cores. Based on the core count, two or one of the CCDs attach to the same switch within the I/O die network. Each of the switches on the I/O die that connects the CCDs also attaches a memory controller with two memory channels, which can result in four non-uniform memory access (NUMA) nodes.

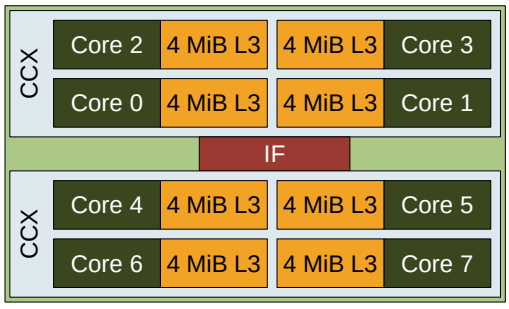
Each core has a common front-end which fetches instructions for two independent hardware threads [17]. The fetch window is 32 B wide and fed to a 4-way decoder. The back-end is split into two parts: One part comprises four Arithmetic Logical Units and three Address Generation Units (AGUs), the other contains two 256-bit-wide Floating-point Multiply-Add (FMA) and two 256-bit-wide floating-point add units. The AGUs can be used for two loads and one store per cycle, where each of these can transfer up to 32 B of data.

Each processor core holds an op cache for 4096 ops, 32 KiB L1I and L1D caches, and 512 KiB L2 cache, which is used for instructions and data. In addition, each CCX holds 16 MiB of L3 cache, distributed over four slices with 4 MiB each.

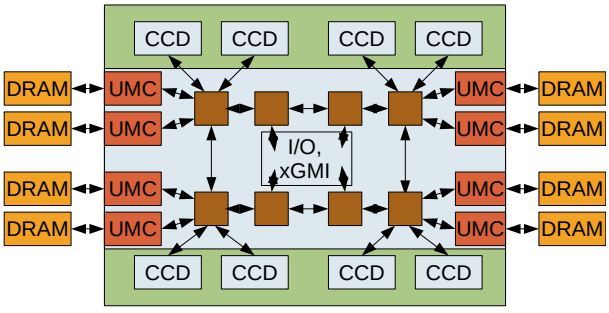
B. Energy Efficiency Details for ACPI States

The AMD Zen 2 architecture implements a wide range of power saving mechanisms. According to AMD's *Processor Programming Reference* [2, Section 2.1.14.3], a maximum of eight P-states can be defined. However, on most systems, the number of available P-states will be lower. The actual number can be retrieved by polling the **P-state current limit** Model Specific Register (MSR). The definition of single P-states includes specifications for frequency, the *"expected maximum current dissipation of a single core"*, and a *"voltage ID"*. The latter is not publicly documented. A processor core frequency can be higher than nominal when using *Core Performance Boost*. No implementation details are disclosed for server architectures. For desktop processors, AMD describes Precision Boost, where the frequency can be increased in 25 MHz steps as part of the SenseMI technology[1]. This would match the frequency multiplier entry in the MSR, where multiples of 25 MHz can be defined.

[1] https://community.amd.com/t5/blogs/understanding-precision-boost-2-in-amd-sensemi-technology/ba-p/416073



(a) Core Complex Die (CCD) with Core Complexes (CCXs)



(b) I/O die with memory controllers (UMC), attached memory, IF-Switches (brown), CCDs, repeaters, and I/O; xGMI attachements not depicted; based on [2]

Fig. 2: Block diagram of AMD Rome Architecture, communication via Infinity Fabric (IF)

Zen 2 implements the usage of idle power states with the instructions `monitor/mwait` and I/O addresses, which when accessed trigger the entering of one of these states. The latter are defined in the **C-state base address** MSR [2, Section 2.1.14.3] . There is no indication that AMD implements clock modulation as Intel does [7]. However, according to Singh et al., processors support more frequency ranges on some market segments with run-time duty-cycle settings [18, Fig. 2.1.5].

C. Other Energy Efficiency Details

In addition to traditional power saving mechanisms, AMD also implements I/O die P-states. According to [17, Section "ROME" SOC], the frequency is decoupled from core P-states and can be used to control the performance and power budget of the I/O die. The reference document [2, Section 2.1.14.3] also indicates that the L3 cache has a dedicated frequency domain and names some restrictions (*"L3 frequencies below* $400\,MHz$ *are not supported by the architecture"*). However, the underlying mechanism is not disclosed.

Parts of the core can be clock-gated at a fine granularity even during active states. Singh et al. state in [18] that *"clock gating opportunities were identified for low-IPC patterns, where only a portion of the pipeline is used"*. Suggs et al. name *"continuous clock and data gating improvements"* in [17, Section Energy Efficiency]. In particular, the upper $128\,\mathrm{bit}$ of the SIMD-capable Floating-Point (FP) units were target for optimization, since only specialized software uses 256-bit SIMD instructions. Singh et al. state that *"Zen 2 gated the FP clock mesh* 128*-bit regions with no additional clocking overhead [...]"*, which *"saved 15% clock mesh power in idle and average application cases where FP was inactive"*. Due to the large number of partially clock-gated, wide super scalar execution units, power consumption now depends on the actual workload being executed. *"[A]n intelligent EDC manager which monitors activity [...] and throttles execution only when necessary"* helps to avoid peaks that *"cause electrical design current (EDC) specifications to be exceeded"* [17, Section Floating-Point/Vector Execute]. We evaluate this in Section V-E.

Burd et al. describe more power saving mechanisms for AMD Zen/Zeppelin processors in [19], which could also be available for AMD Zen 2/Rome. These include a package C-state PC6 *"in which the CPU power plane can be brought to a low voltage when there are no active CPU cores"*, but also a low power state that could be implemented in the I/O die. In this state, *"most of the IO and memory interfaces are disabled and placed in a low-power state"*. Burd et al. also mention the possibility to lower the infinity fabric link width between sockets.

With the Zen architecture, AMD replaced APM (Application Power Management) with RAPL (Running Average Power Limit) [2]. The implementation seems similar to the Intel solution, but uses different MSRs. While Intel typically provides multiple domains and the option to limit power consumption over various time frames [20, Section 14.10], AMD only describes registers for reading package and core power consumption. However, the latter is available with a per-core spatial resolution, compared to per-package for Intels core domain (pp0). While Intel switched from a model to measurement with the Haswell architecture, slides from AMD indicate that they use a model based on *"*>1300 *critical path monitors,* 48 *on-die high speed power supply monitors,* 20 *thermal diodes, [and]* 9 *high speed droop detectors"*[2] for Zen desktop processors. We evaluate the resulting accuracy of RAPL in Section VII.

In [19], Burd et al. describe how on the Zen architecture, System Management Units (SMUs) work together to communicate applied frequencies and necessary voltages, where each die of the package implements its own SMU. In [19, Fig. 7], they describe that from this set of SMUs, a Master SMU is chosen, which evaluates data from other SMUs and runs control loops for package power and temperature. It also triggers frequency changes and controls the external voltage regulator. The slide set of [21][3] also shows SMUs being part of the Rome architecture and still responsible for *"power management"* and *"thermal control"*.

IV. Test System and Power Measurements

For our analysis, we use a dual socket system with two AMD EPYC 7502 processors, where each processor hosts 32 Cores in 4 CCDs. We configured the system to use the *"2-Channel Interleaving (per Quadrant)"* mode [22]. From the available frequencies ($1.5\,\mathrm{GHz}$, $2.2\,\mathrm{GHz}$ and $2.5\,\mathrm{GHz}$), we use the reference frequency ($2.5\,\mathrm{GHz}$) and the "Auto" I/O die P-state unless specified otherwise. By default, memory is clocked at $1.6\,\mathrm{GHz}$. The system runs Ubuntu Linux 18.04 with kernel 5.4.0-47-generic. We use the GNU Compiler Collection (GCC) in version 7.5.0 as the default compiler. Access to MSRs is performed via the *msr* kernel module, except for RAPL energy readouts, for which we use custom libraries[4]. We use the Linux cpufreq governor "userspace" to control processor frequencies. By default, we enabled all available C-states. We use `sysfs` files to control C-states[5] and hardware threads[6].

We use a ZES LMG670 power analyzer with L60-CH-A1 channels to measure the total AC power consumption of the test system. In our configuration, the power measurement has an accuracy of $\pm(0.015\,\% + 0.0625\,\mathrm{W})$. During the experiments, a separate system collects the active power values at $20\,\mathrm{Sa/s}$. The out-of-band data collection avoids any perturbation. Measurement data is merged with the internal power and performance monitoring in a post-mortem step. For quantitative comparisons, we use average power values within the inner $8\,\mathrm{s}$ of a $10\,\mathrm{s}$ interval in which one workload configuration is executed continuously. This approach avoids inaccuracies due to misaligned timestamps. We pre-heat the system for power-sensitive workloads.

[2]Michael Clark, The "Zen" Architecture, https://www.slideshare.net/pertonas/amd-ryzen-cpu-zen-cores-architecture

[3]https://www.slideshare.net/AMD/amd-chiplet-architecture-for-highperformance-server-and-desktop-products

[4]https://github.com/tud-zih-energy/x86_energy with `x86_adapt` backend

[5]`/sys/devices/system/cpu/cpu\d+/cpuidle/state[012]`

[6]`/sys/devices/system/cpu/cpu\d+/online`

V. Processor Frequencies

A. Influence of Idling Hardware Threads on Core Frequencies

To investigate the influence of idling hardware threads, we set up a simple workload, where one thread of a core executes a constant workload (`while(1);`) running at minimum frequency (1.5 GHz). We monitor its frequency with `perf stat -e cycles -I 1000`. Then, we change the frequency of the second thread of the same core to the nominal frequency (2.5 GHz). We let the second thread idle and monitor its activity also with `perf stat`. The idling thread reports only a usage of less than 60 000 cycle/s and uses idling states. However, even though the second thread is idling, the frequency of the first thread is elevated to the nominal frequency (2.5 GHz) rather than its configured minimal frequency (1.5 GHz). In another attempt, we disable the idling thread. Still, the frequency of the core is defined by the offline thread. Based on this observation, it can be advantageous to set the frequency of unused hardware threads to the minimal frequency to allow active threads to control their effective frequency. We never observed this behavior on Intel processors with enabled deep idle states. It may therefore be unexpected for system administrators.

B. Frequency Transition Delays

While operating systems change processor frequencies based on resource usage, researchers also use these mechanisms to optimize energy efficiency for code paths [23], [24]. However, the possible time scales for both highly depend on the delay of the frequency transition, which can take tens to hundreds of microseconds [4], [12].

For our tests, we refined the approach from [4] to measure this delay as follows: The benchmark switches the core frequency and measures the runtime of a minimal workload. It repeats the measurement until the expected performance of the target frequency is reached. Afterwards, the performance is measured another 100 times and validated with a confidence interval of 95 %. Then, the benchmark applies the initial frequency, waits for the appropriate performance level and validates it as before. If either one of the two validations fails, the sample and the following sample is discarded. Before the next measurement, the benchmark waits a random time between 0 ms and 10 ms. We measure each combination of initial and target frequency 100 000 times. Other cores in the system are set to the minimum frequency of 1.5 GHz.

Figure 3 shows the distribution of transition delays for a switch from 2.1 GHz to 1.5 GHz. The measured latencies are approximately uniformly distributed between 390 µs and 1390 µs. This wide distribution indicates that an internal fixed update interval of 1 ms is used. A similar mechanism was observed for core and uncore frequencies of Intel processors [12], [16]. The delay from the initial request to a transition slot is up to 1 ms and the actual frequency change takes another 390 µs.

We also observe comparable results for other frequency combinations with the exception of changes between 2.5 GHz and 2.2 GHz, where we experienced a significantly higher

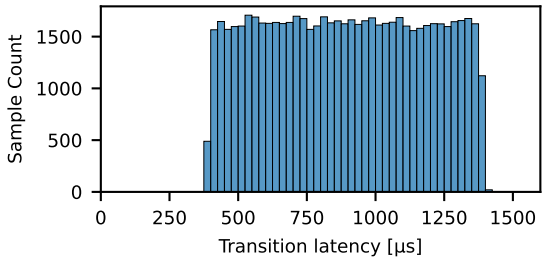


Fig. 3: Histogram of frequency transition delays from 2.2 to 1.5 GHz (random starting time, 25 µs bins).

rate of invalid measurements. When switching from 2.5 GHz to 2.2 GHz, some measured latencies are below the assumed minimal transition delay, down to 160 µs. Symmetrically, there are less measurement samples above 1100 µs. When switching from 2.2 GHz to 2.5 GHz, some transitions are executed instantaneously (1 µs delay). Here, the previous transition did not finish completely (e.g., frequency set, but not voltage). Therefore, returning to a previous setting is faster. The effect disappears with random wait times of at least 5 ms.

Based on the measurements, we can show that AMD introduced update intervals for core frequencies that define times when frequency transitions can be initiated. On our system, the period of that window is 1 ms, compared to 500 µs on Intel systems [12], [16]. The delay of approx. 390 µs for the actual transition (360 µs for increasing frequency) is also significantly higher compared to the Intel Haswell architecture (21 µs to 24 µs). This might be caused by communication between the SMUs, which likely creates higher delays compared to a centralized Power Control Unit on Intel architectures.

C. Influence of Mixed Frequencies on a Single CCX

For this evaluation, we configure cores of a single CCX to use different frequencies. We run a simple workload (`while(1);`) on all cores of a CCX and measure the frequency of one core, which is configured differently than other cores. We monitor each setup for 120 s and capture the frequency every second via `perf stat`. The results are presented in Table I. Evidently, core frequencies are reduced if other cores on the same CCX apply higher frequencies. While this effect is moderate for a core running on 1.5 GHz, with a reduction of 33 MHz and 71 MHz, the performance penalty for 2.2 GHz is more severe with a reduction of 200 MHz.

To understand the influence different core frequencies have on the L3-cache frequency, we use a pointer chasing benchmark, developed by Molka et al. [25]. We disabled hardware prefetchers in this test and explicitly used huge pages via the *hugetlbfs*. As with the previous test, we test one core of one CCX, while the other cores are in an active state. We measure each combination several times and present the minimal measured latencies to filter out outliers, where the measurement has been influenced by software (OS) or hardware (processor internal mechanisms). As Figure 4 shows, the latency to the L3 cache decreases for a core running at 1.5 GHz, when other cores in the same CCX apply a higher

TABLE I: Applied mean core frequencies GHz in a mixed frequency set-up on one CCX, Tests with lower applied frequency than other cores are highlighted.

		Set frequencies of other cores [GHz]		
		1.5	2.2	2.5
Set frequency of measured core [GHz]	1.5	1.499	**1.466**	**1.428**
	2.2	2.200	2.199	**2.000**
	2.5	2.497	2.499	2.499

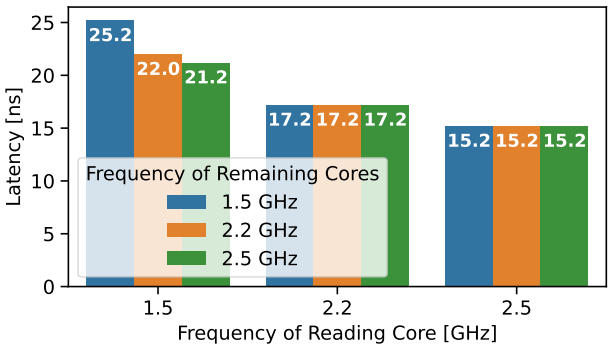


Fig. 4: L3-cache latencies in a mixed frequency set-up on one CCX.

core frequency, even though its own frequency is decreased by the previously mentioned effect. We explain this with an increased L3-cache frequency that is defined by the highest clocked core in the CCX.

Both effects have severe consequences for performance modeling and energy efficiency optimizations. Even if an optimal core frequency is predicted correctly and applied to a processor core, other cores can disturb the well-optimized setup, resulting in a loss of performance and energy efficiency. However, the same mechanism that can reduce the frequency and subsequently performance of a single core in a CCX can also decrease L3-cache latencies and therefore increase performance.

D. Influence of I/O Die P-state and DRAM Frequency on Memory Performance

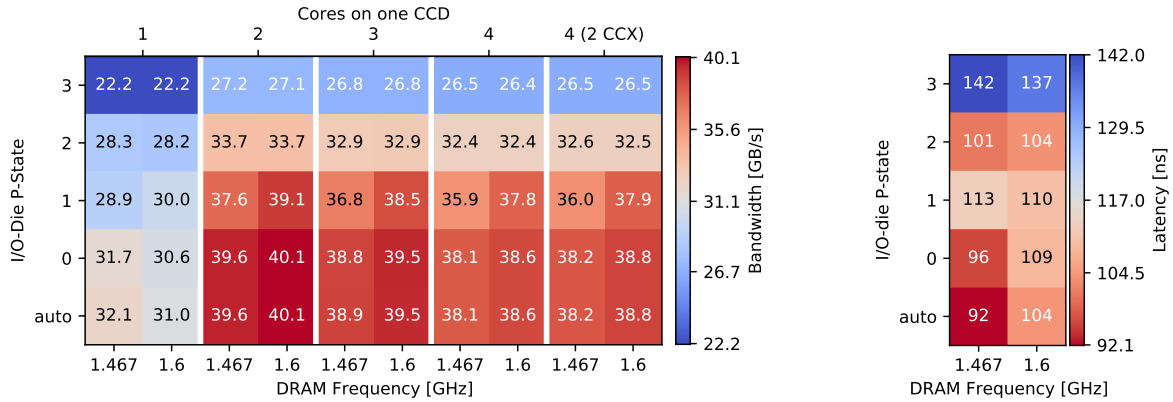
In addition to core and L3-cache frequencies, the I/O die has its own voltage and frequency domain, which can influence NUMA, I/O, and memory accesses that pass the I/O die. In this section, we analyze how different configurations for I/O die and memory frequencies will influence main memory performance. To do so, we use two benchmarks: The STREAM-Triad benchmark proposed by McCalpin [26], and the memory latency benchmark described by Molka et al. [25]. In contrast to previous measurements, we use the Intel compiler for the STREAM benchmark as it reaches higher performance levels. We further vary the number of cores that concurrently access memory by using additional well placed threads, defined via OpenMP environment variables. We disabled hardware prefetchers, as mentioned in Section V-C, and explicitly use huge pages for the latency benchmark. For both benchmarks we vary I/O die P-states and DRAM frequencies in the BIOS.

Figure 5 presents the results. The pattern for memory bandwidths shows that two cores on one CCX already reach the maximal main memory bandwidth and additional cores can lead to performance degradation. Using higher I/O die P-states reduces power consumption but also lowers memory bandwidth. Surprisingly, a higher DRAM frequency does not increase memory bandwidth significantly. As expected, the lowest power state performs best in this benchmark and the auto setting has the same performance. This could lead to the assumption that for performance analysis it would be best to pin the I/O die P-state to 0 to remove one source of inpredictability. However, when looking at the latencies-result, one can see that auto outperforms the P-state 0 with 92.0 ns vs 96.0 ns. Moreover, for the higher memory frequency, also the I/O die P-state 2 performs better than P-state 0. This could be attributed to a better match between the frequency domains for memory and I/O die. According to our observations, the auto setting performs good for all scenarios. However, we did not investigate the hardware control loop and how fast it reacts to different access patterns.

E. Frequency Limitations for High-Throughput Workloads

Starting with the Haswell-EP processor generation, Intel defined AVX frequencies, where workloads that use wide SIMD instructions would use a lower "nominal" frequency [10]. [17, Section Floating-Point/Vector Execute] describes such a static assignment a *"simplistic approach to mitigating this issue"* and describes that Zen 2 uses *"an intelligent EDC manager which monitors activity [...] and throttles execution only when necessary"*. To evaluate how the test system is influenced by workloads that utilize all processor resources to the highest extend, we use FIRESTARTER 2 [27]. The workload schedules up to two 256-bit FMA instructions per cycle accompanied by 256-bit vector loads and stores to different levels of the memory hierarchy. To maximize back-end utilization, these instructions are interleaved with integer and logical instructions. To utilize the front-end, we increase the size of the inner loop such that it does not fit into the L0 op cache but in the L1I cache. This limits the maximal throughput of each core to four instructions per cycle. Before we run our tests, we execute FIRESTARTER for 15 min in order to create a stable temperature. We run our tests at nominal frequency for two minutes and measure frequency and throughput with `perf stat`. For power measurements, we use the external AC measurements described in Section IV as well as RAPL package energy counters. We exclude data for the first 5 s and last 2 s to avoid including the initialization phase in addition to clock synchronization issues. Performance data for all used threads is collected in 1 s intervals.

In our measurements, average processor core frequencies are reduced to 2.0 GHz or 2.1 GHz, depending on whether two threads per core are used or not. This is depicted in Figure 6. The standard deviation is 3.04 MHz and 0.82 MHz, respectively. The frequency difference can be explained with a higher average throughput of 3.56 $\frac{\text{instruction}}{\text{core cycle}}$ (standard deviation 0.008) instead of 3.23 $\frac{\text{instruction}}{\text{core cycle}}$ (standard deviation



(a) STREAM Triad bandwidth

(b) latency

Fig. 5: DRAM bandwidth and latency for I/O die P-states and DRAM frequencies.

0.004), respectively. Also the average power consumption of the system is higher when both hardware threads of a core are used. It is 509 W instead of 489 W with a standard deviation of less than 4 W in both cases. Meanwhile, the RAPL package counter reports 170 W for both processors even though their TDP is stated to be 180 W. This might be related to the accuracy of the processor internal power measurement, which we investigate in Section VII. Enabling Core Performance Boost has almost no influence on throughput, frequency and power consumption.

Based on our measurements, we conclude that the EDC manager works as expected and will lower processor frequencies if needed. This poses a threat to the efficiency of HPC systems. In well balanced applications, one throttling processor can slow down the whole program. On Intel systems, administrators and users know about the severity of this problem and can apply counter measures (e.g., running highly parallel programs at reduced frequency) based on the documented AVX frequency ranges. For AMD Rome systems, measurements are required to determine the actual frequency ranges on a specific processor.

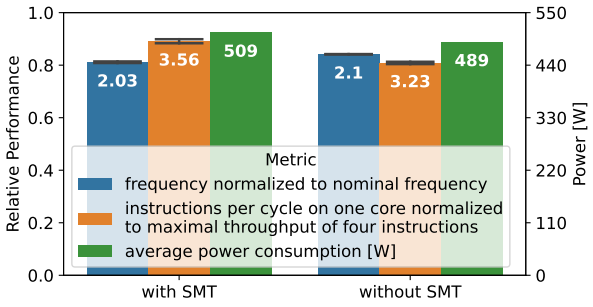


Fig. 6: Observed parameters at set nominal frequency 2.5 GHz) with and without the usage of two hardware threads per core. Error bars depict standard deviation.

VI. Power State Details

In this section, we analyze processor power states (C-states), as described in [3, Section 8.6]. Idling C-states are triggered by the operating system when there is no thread that can be scheduled on a hardware thread. When all threads of a processor core enter such a state, (parts of) the processor core can be clock gated [28, Section 5.2.1.1] or even power gated [28, Section 5.3.2]. When all cores of a processor are in an idle power state, the processor can take additional measures, e.g., clock and power gating shared components, which lowers the power envelop of the processor further. However, returning from an idling power state takes some time, which can violate the requirements for real-time systems. To support the decision of the OS, which C-state to use, processors typically hand over ACPI objects describing the transition latency and the average power consumption [3, Section 8.4.2.1]. On our test system, three C-states are supported, the active C-state C0, and two additional idling C-states, C1 and C2[7]. While the former is entered with the instructions `monitor` and `mwait`, the latter uses IO address 0x814 in the C-state address range described in Section III-B. Transition times are reported as 1 µs and 400 µs, respectively. The power values reported by the hardware to the OS are `UINT_MAX` for the active C-state and `0` for the idle states and cannot contribute towards an informed selection of C-states.

A. Power Consumption in Different Power States

In the following, we characterize the average AC power consumption of the full system in different configuration of idle states, each measured for 10 s. All configurations are shown in Figure 7. When all hardware threads are using the C2 state to the extent that is possible on a standard Linux system with regular interrupts, the system consumes 99.1 W. In the following experiment, we put more hardware threads in C1 by disabling the C2 state in `sysfs`. The change is applied linearly, following the logical CPU numbering in steps of single CPUs. We start with the hardware thread of each

[7]This paper uses the OS C-state numbering.

core within the first processor package, the second processor package, and then the second hardware threads of each core, again grouped by package. With a single core using C1 rather than C2, the power consumption increases by 81.2 W to a total of 180.3 W. Additional cores in C1 only increase power consumption by 0.09 W each with no further change when the second hardware thread of each core is put into C1. The minor frequency-independent additional power per core is consistent with the observation that the hardware counters for `cycles`, `aperf`, and `mperf` do not advance on cores that are in C1. Both effects indicate that cores are clock-gated during C1.

For the active state (C0) we pin an unrolled loop of `pause` instructions to each hardware thread. This workload exhibits a more stable and slightly lower power consumption than `POLL`, which is also based on `pause`, but without unrolling and more sophisticated checks for each iteration. With one active thread and all others in C2, the system uses almost the same power (180.4 W) than with one thread in C1 and all others in C2. While C1 and C2 power was independent of core frequency, active power does depend on frequency as expected. For 2.5 GHz, additional active cores increase power by 0.33 W each and hardware thread costs 0.05 W each. On our dual-socket system, there was no measurable impact of activating the second package. There appears to be only one criterion for deep package sleep states: All threads of all packages must be in the deepest sleep state. The C1 state is only relevant for one specific core, as opposed to the C1E state on Intel systems.

The reported numbers are only valid for our specific system and depend on the processor model, processor variations, and other components. However, this example particularly highlights the disproportionately high cost of not using the deepest sleep states on a single hardware thread and thus the importance of managing C-states correctly on idle systems.

Compared with a dual socket Intel system using Xeon Gold 6154 CPUs [16, Section III], the deepest idle state (69 W, all C6) and first core in C1E (+97 W) are in a similar order of magnitude. However, on the Intel Skylake system, each additional active core (`pause` loop) costs 3.5 W - about ten times the power of our AMD Rome system.

B. Influence of Idling Hardware Threads on Idle States

In some scenarios, administrators disable the additional hardware threads of each core via operating system interfaces, to decrease the probability of leaving a package C-state and subsequently increasing the average power consumption during idle times. While such an optimization can be recommended on Intel systems, we would strongly discourage using this option on AMD Rome. Under conditions we could not yet clearly identify, a strange behavior was observable: even though C2 states are active and used by the active hardware threads, system power consumption is increased to the C1 level as long as the disabled hardware threads are offline. Only an explicit enabling of the disabled threads will fix this behavior. While we cannot pinpoint it to either Linux OS or AMD processor, we assume that it is the interaction between both, elevating some disabled hardware threads to C1.

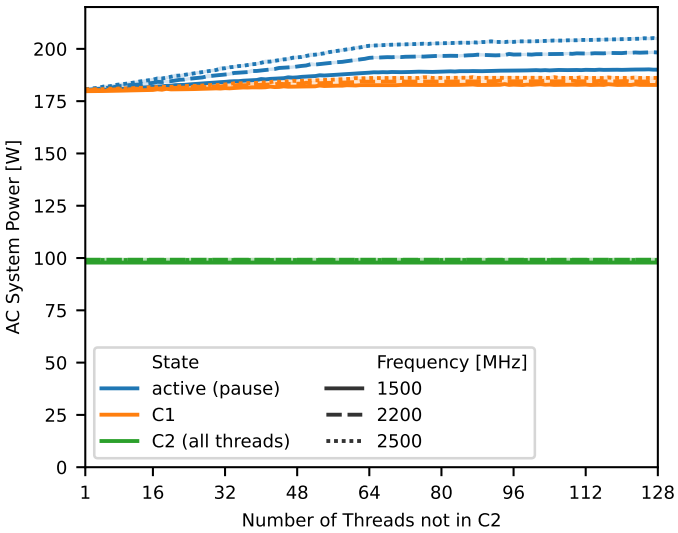


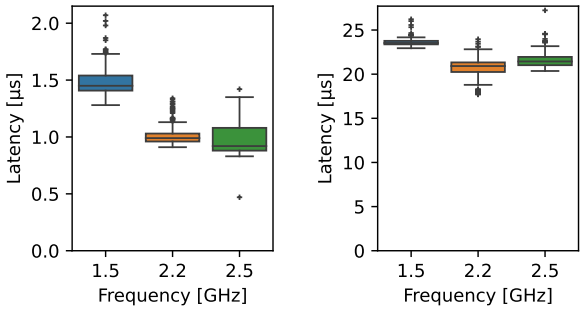
Fig. 7: Average full system AC power consumption for different idle combinations with increasing number of hardware threads in lower C-states.

C. Power State Transition Times

To determine the transition times for idle states, we use the workload Ilsche et al. explained in [6]. However, we had to change the logged event for triggering the transition to *sched_waking*, since the newer Linux kernel does not report *sched_wake_idle_without_ipi*, which Ilsche et al. used for the test case. The remaining setup stays the same: Two threads are started: caller and callee. While the callee is idling via `pthread_cond_wait`, the caller wakes it with `pthread_cond_signal`. We schedule the threads within a CCX for local measurements and on one core of each of the two sockets for remote measurements. We also measure the influence of frequencies and take 200 samples for each combination of C-state, frequency, and local/remote-scenario. Results for local transition times are depicted in Figure 8.

The outliers can be attributed to the measurement, which runs on the same resources as the test workload and therefore influences the results. Also, the depicted C-state is the one requested by the OS, not necessarily the one that is realized by the hardware. The latency for returning from C1 is consistent with the value reported by hardware with ∼1 µs at 2.2 GHz and 2.5 GHz and 1.5 µs at 1.5 GHz. The C2 latency is between 20 µs and 25 µs and significantly lower than reported to the OS (400 µs). However, this value could significantly increase when package C-states are used, which disable additional processor components. This case is not measurable with the used methodology since the active caller would prevent package C-states.

Transition times for remote configurations only add a small overhead (∼1 µs) to the results shown in these diagrams and are therefore not presented in this paper. However, this validates the finding from Section VI-A: package C-states are not used as long as a single core (which runs the caller thread) in the system is active.



(a) C1 (b) C2

Fig. 8: C-state transitions (caller and callee in same CCX).

VII. Integrated Energy Measurement with RAPL

In this section, we analyze AMD's RAPL implementation. We separately analyze its quality with a high-level focus on the executed workload and the detailed impact of input data on instruction-level power consumption. We measured an update rate of 1 ms for RAPL by polling the MSRs via the `msr` kernel module, which meets the specification for Intel processors.

A. Quality of the Integrated Power Measurement

To analyze the accuracy of RAPL readings, we follow the methodology presented by Hackenberg et al. [12]. We execute a set of experiments, where each forms a particular combination of workload, thread placement, frequency, and enabled C-states, for a duration of 10 s. We record RAPL package energy measurements, RAPL core information, and system AC power for each workload configuration as described in Section IV.

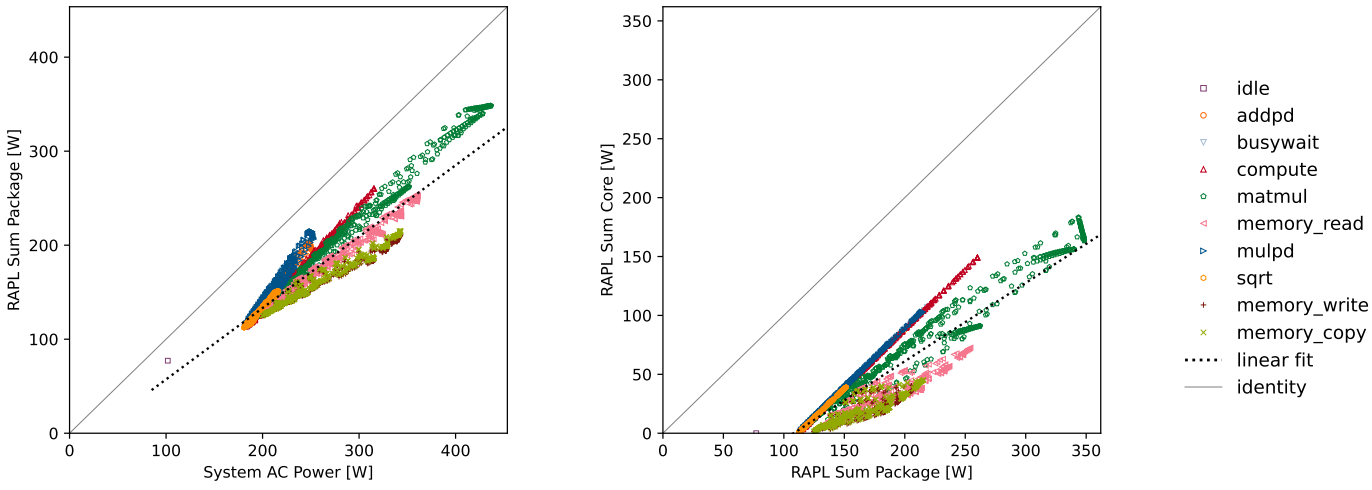
If AMD RAPL would use an accurate measurement that covers all components for which power varies by workload, a single function would map RAPL readings to the reference measurement. Instead, Figure 9a is reminiscent of Intel's implementation before Haswell [11]. The results indicate that the energy data is modeled, not measured: Even workloads that do not use memory (`sqrt`, `add_pd`, `mul_pd`) show inconsistent power. Further, the energy consumption of memory accesses (e.g., `memory_read`, `memory_write`) is not fully captured by RAPL. No DRAM domain is available and the RAPL package domain reports significantly lower power compared to the external measurement. Considering the different domains, this does not necessarily imply that RAPL readouts are wrong. But it shows that they cannot be used to accurately estimate and therefore optimize for total system power, as opposed to Intel systems since Haswell. On such systems, this is possible when adding Package and DRAM energy [12].

The comparison in Figure 9b reveals that there is a simple relation between the different RAPL measurement domains for compute-only workloads while the power difference for memory-intensive workloads and idle varies. This seems intuitive, since a model for package power consumption would include modeled core energy but also shared non-core resources.

B. Measurement of Data-dependent Power Consumption

The power consumption for executing a workload does not only depend on the used instructions, but also on the processed data. Given data-dependent power differences of up to ∼15 % full system power [16], data can also have a significant impact on RAPL accuracy. Subsequently, applied processor frequencies can be inaccurate, which can lead to an exceeded TDP or performance loss.

Another aspect to consider is the exploitation of RAPL for software-based side-channel attacks as demonstrated on Intel systems. In [14], Lipp et al. also indicate that it could be possible on newer AMD systems. On the one hand, the authors distinguish different *instructions* based on RAPL measurements. While not always accurate, Figure 9 confirms that RAPL on Zen 2 does reflect the different power consumption of instructions to some extent. On the other hand, Lipp et al. use RAPL to distinguish *operands* of instructions. To that end,



(a) Reference vs. RAPL Package (b) RAPL core vs. RAPL package

Fig. 9: Readings of RAPL on AMD Epyc 7502 and the AC reference measurements in relation to each other.

they measure the energy consumption of the `shr` instruction with RAPL on an AMD Zen 2 desktop system and show a slightly shifted probability density function for different numbers of set bits (operand Hamming weight).

We measure the instruction power consumption by repeating an unrolled loop of the respective instruction for a fixed number of total instructions. Successive instructions use different registers to avoid stalling. For each block of instructions, the test application randomly chooses a relative Hamming weight of either 0, 0.5, or 1 and executes this configuration on all hardware threads. The instruction count is chosen such that each instruction block runs for 10 s. Overall, 3000 instruction blocks are executed (∼1000 per operand weight). The experiment application collects RAPL energy values between instruction blocks. Even though the measured duration is very stable, we normalize the energy values to power.

First, we look at a 256-bit `vxorps` instruction and vary the the operand that determines the toggled bits in the destination register. Figure 10 illustrates the distribution of average power values for the repeated instruction blocks of each operand configuration. To avoid smoothing, we use empirical cumulative distribution plots. Moreover, to confirm whether the distribution is stable, we separate the samples into ten random subsets and plot the distribution for each subset. As can be seen in Figure 10a, the system power consumption increases with the number of toggled bits with a significant difference of 21 W (7.6 %) with no overlap in distributions. The RAPL measurements do not reflect this difference: Their overall averages are within 0.08 % for different operand weights. Figure 10b shows that the distributions are distinguishable but strongly overlapping. Moreover, the clear ordering between operand weights 0, 0.5, 1 is not reflected by RAPL.

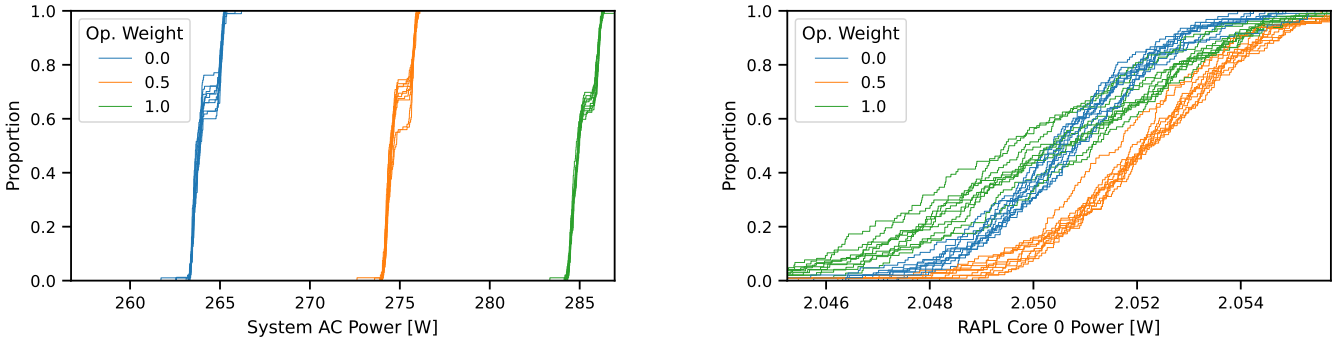
To contrast the findings of [14], we also ran the experiment with a 64-bit `shr` instruction. The operand is seeded depending on the selected operand weight and repeatedly shifted by 0. The system power consumption averages are much closer within 0.9 % for different operand weights whereas RAPL core power averages are within 0.015 % and their distribution overlaps similarly to the previous experiment. In all cases, the RAPL package domain measurements behave similarly, albeit with different ordering of distributions.

Primarily, the results show that the RAPL implementation on our system does not correctly represent the impact of data on power consumption, possibly affecting measurement accuracy in workloads with biased data. However, it is conceivable that this RAPL implementation could still be used to leak information about the processed data through very small differences in the distribution of power consumption samples. The results indicate that this is due to indirect effects, e.g., an increased temperature based on the number of set bits. Nevertheless, distinguishing the operand weight from RAPL values on this system would take substantially more samples compared to a physical measurement. Moreover, on our test system, RAPL is not accessible to unprivileged users.

VIII. Conclusion and Future Work

With the Zen 2 Rome processors, AMD ships a complex x86 architecture, which includes numerous power saving and monitoring mechanisms for an improved energy efficiency. In this paper, we provide a detailed analysis of these.

To maximize energy efficiency, users and administrators can take the following measures: Hardware threads should not be disabled in the OS as this can disable package C-states and significantly increase idle power consumption under specific circumstances. Unused hardware threads should be run at the lowest possible frequency. Otherwise, they can raise the frequency of hardware threads on the same core. Mixed frequencies within one CCX should be avoided, since this can lead to performance losses on cores with lower frequency settings. The processor frequency should be monitored to detect throttling when using 256-bit SIMD instructions. This can lead to significant performance degradation, especially for highly parallel HPC codes. Energy measurements of AMDs RAPL implementation should be considered inaccurate. No DRAM domain is provided, and DRAM energy consumption is not (fully) included in the package domain. Therefore, AMD's



(a) Reference measurement, full system

(b) RAPL, core

Fig. 10: Full system AC and RAPL power consumptions for `vxorps`. Each chart shows the empirical cumulative distribution of ten random sample sets for the three different relative operand Hamming weights.

RAPL is unsuitable to optimize total energy consumption. The modeled approach also fails on reflecting the influence of operands, which can also be seen as a benefit when it comes to power measurement based side-channel attacks.

Our findings are valuable for a wide audience: Performance models can be improved for a better accuracy, tuning mechanisms can be refined to become more efficient, and operating systems can be optimized to fix or prevent some of the identified peculiarities.

As future work, we will analyze the frequency throttling on processors with more cores. We expect a more severe impact, since the ratio of compute to I/O resources is higher. We will also analyze the memory architecture and the influence of power saving mechanisms on these in higher detail. Finally, we plan to analyze why offline hardware threads can prevent the usage of package C-states.

Acknowledgments and Reproducibility

This work is supported in part by the German Research Foundation (DFG) within the CRC 912 - HAEC.

Measurement programs, raw data, and chart notebooks are available at https://github.com/tud-zih-energy/2021-rome-ee.

References

[1] E. Strohmaier, J. Dongarra, H. Simon, M. Meuer, and H. Meuer, "Top500," 2020, https://top500.org (accessed 2020-11-29).

[2] Advanced Micro Devices, "Preliminary Processor Programming Reference (PPR) for AMD Family 17h Model 31h, Revision B0 Processors," 2020, https://developer.amd.com/wp-content/resources/55803_B0_PUB_0_91.pdf(accessed 2020-11-18).

[3] "Advanced Configuration and Power Interface (ACPI) specification, revision 6.3," Unified Extensible Firmware Interface (UEFI) Forum, Inc., Jan. 2019, https://uefi.org/sites/default/files/resources/ACPI_6_3_final_Jan30.pdf (accessed 2020-11-18).

[4] A. Mazouz, A. Laurent, B. Pradelle, and W. Jalby, "Evaluation of CPU Frequency Transition Latency," *Computer Science - Research and Development*, 2014, DOI: 10.1007/s00450-013-0240-x.

[5] R. Schöne, D. Molka, and M. Werner, "Wake-up Latencies for Processor Idle States on Current x86 Processors," *Computer Science - Research and Development*, 2014, DOI: 10.1007/s00450-014-0270-z.

[6] T. Ilsche, R. Schöne, P. Joram, M. Bielert, and A. Gocht, "System Monitoring with lo2s: Power and Runtime Impact of C-State Transitions," in *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2018, DOI: 10.1109/IPDPSW.2018.00114.

[7] R. Schöne, T. Ilsche, M. Bielert, D. Molka, and D. Hackenberg, "Software Controlled Clock Modulation for Energy Efficiency Optimization on Intel Processors," in *Proceedings of the 4th International Workshop on Energy Efficient Supercomputing (E2SC)*, ser. E2SC '16. IEEE Press, 2016, DOI: 10.1109/E2SC.2016.15.

[8] Advanced Micro Devices, "Processor Programming Reference (PPR) for AMD Family 17h Model 60h, Revision A1 Processors," 2020, https://www.amd.com/system/files/TechDocs/55922-A1-PUB.zip(accessed 2021-06-18).

[9] B. Rountree, D. H. Ahn, B. R. de Supinski, D. K. Lowenthal, and M. Schulz, "Beyond DVFS: A First Look at Performance under a Hardware-Enforced Power Bound," in *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops PhD Forum*, 2012, DOI: 10.1109/IPDPSW.2012.116.

[11] D. Hackenberg, T. Ilsche, R. Schöne, D. Molka, M. Schmidt, and W. E. Nagel, "Power Measurement Techniques on Standard Compute Nodes: A Quantitative Comparison," in *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2013, DOI: 10.1109/ISPASS.2013.6557170.

[10] J. Schuchart, D. Hackenberg, R. Schöne, T. Ilsche, R. Nagappan, and M. K. Patterson, "The Shift from Processor Power Consumption to Performance Variations: Fundamental Implications at Scale," *Computer Science - Research and Development*, 2016, DOI: 10.1007/s00450-016-0327-2.

[12] D. Hackenberg, R. Schöne, T. Ilsche, D. Molka, J. Schuchart, and R. Geyer, "An Energy Efficiency Feature Survey of the Intel Haswell Processor," in *2015 IEEE International Parallel and Distributed Processing Symposium Workshop*. IEEE, 2015, DOI: 10.1109/ipdpsw.2015.70.

[13] M. Hähnel, B. Döbel, M. Völp, and H. Härtig, "Measuring Energy Consumption for Short Code Paths Using RAPL," *SIGMETRICS Perform. Eval. Rev.*, Jan. 2012, DOI: 10.1145/2425248.2425252.

[14] M. Lipp, A. Kogler, D. Oswald, M. Schwarz, C. Easdon, C. Canella, and D. Gruss, "PLATYPUS: Software-based Power Side-Channel Attacks on x86," in *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2021.

[15] C. Gough, I. Steiner, J. Koomey, and L. Cheng, *Energy Efficient Servers: Blueprints for Data Center Optimization*. Apress Media, 2015, ISBN: 978-1-4302-6638-9.

[16] R. Schöne, T. Ilsche, M. Bielert, A. Gocht, and D. Hackenberg, "Energy Efficiency Features of the Intel Skylake-SP Processor and Their Impact on Performance," in *2019 International Conference on High Performance Computing Simulation (HPCS)*, 2019, DOI: 10.1109/HPCS48598.2019.9188239.

[17] D. Suggs, M. Subramony, and D. Bouvier, "The AMD "Zen 2" Processor," *IEEE Micro*, 2020, DOI: 10.1109/MM.2020.2974217.

[18] T. Singh, S. Rangarajan, D. John, R. Schreiber, S. Oliver, R. Seahra, and A. Schaefer, "Zen 2: The AMD 7nm Energy-Efficient High-Performance x86-64 Microprocessor Core," in *2020 IEEE International Solid- State Circuits Conference - (ISSCC)*, 2020, DOI: 10.1109/ISSCC19947.2020.9063113.

[19] T. Burd, N. Beck, S. White, M. Paraschou, N. Kalyanasundharam, G. Donley, A. Smith, L. Hewitt, and S. Naffziger, ""Zeppelin": An SoC for Multichip Architectures," *IEEE Journal of Solid-State Circuits*, 2019, DOI: 10.1109/JSSC.2018.2873584.

[20] *Intel 64 and IA-32 Architectures Software Developer's Manual Volume Volume 3 (3A, 3B, 3C & 3D): System Programming Guide*, Intel, Nov 2020, https://software.intel.com/content/dam/develop/external/us/en/documents-tps/325384-sdm-vol-3abcd.pdf (accessed 2020-11-18).

[21] S. Naffziger, K. Lepak, M. Paraschou, and M. Subramony, "AMD Chiplet Architecture for High-Performance Server and Desktop Products," in *2020 IEEE International Solid- State Circuits Conference - (ISSCC)*, 2020, DOI: 10.1109/ISSCC19947.2020.9063103.

[22] Advanced Micro Devices, "Socket SP3 Platform NUMA Topology for AMD Family 17h Models 30h–3Fh," 2019, https://developer.amd.com/wp-content/resources/56338_1.00_pub.pdf (accessed 2020-11-18).

[23] B. Rountree, D. K. Lowenthal, B. R. de Supinski, M. Schulz, V. W. Freeh, and T. Bletsch, "Adagio: Making DVS Practical for Complex HPC Applications," in *Proceedings of the 23rd International Conference on Supercomputing*, ser. ICS '09. New York, NY, USA: Association for Computing Machinery, 2009, DOI: 10.1145/1542275.1542340.

[24] O. Vysocký, L. Říha, and A. Bartolini, "Application Instrumentation for Performance Analysis and Tuning with Focus on Energy Efficiency," *Concurrency and Computation: Practice and Experience*, 2020, DOI: 10.1002/cpe.5966.

[25] D. Molka, D. Hackenberg, R. Schöne, and M. S. Müller, "Memory Performance and Cache Coherency Effects on an Intel Nehalem Multiprocessor System," in *2009 18th International Conference on Parallel Architectures and Compilation Techniques*, 2009, DOI: 10.1109/PACT.2009.22.

[26] J. D. McCalpin, "Memory Bandwidth and Machine Balance in Current High Performance Computers," *IEEE computer society technical committee on computer architecture (TCCA) newsletter*, vol. 2, pp. 19–25, 1995.

[27] R. Schöne, M. Schmidl, M. Bielert, and D. Hackenberg, "FIRESTARTER 2: Dynamic Code Generation for Processor Stress Tests," in *IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 2021, (accepted).

[28] Weste, Neil H. E. and Harris, David M., *CMOS VLSI Design - A Circuits and Systems Perspective, 4th Edition*. Pearson, 2011, ISBN: 0321547748.