Sleep Well: Pragmatic Analysis of the Idle States of Intel Processors

Till Smejkal*
till.smejkal@tu-dresden.de
TU Dresden
Germany

Jan Bierbaum*
jan.bierbaum@tu-dresden.de
TU Dresden
Germany

Thomas Oberhauser*
thomas.oberhauser@tu-dresden.de
TU Dresden
Germany

Horst Schirmeier
horst.schirmeier@tu-dresden.de
TU Dresden
Germany

Hermann Härtig
hermann.haertig@tu-dresden.de
TU Dresden
Germany

ABSTRACT

Rising energy consumption is of growing concern for cloud data center providers. Modern processors try to counteract this problem through low-power idle states that save energy in phases with little demand for compute resources. Making proper use of this feature, however, requires knowledge about the properties of these states for the very processors used in a specific setup; most importantly, the energy consumed in each idle state and the latency for resuming normal operation. Unfortunately, hardware vendors usually do not provide this critical information.

In this paper, we propose a scheme for automatically analyzing the idle states of modern Intel processors. Our open-source implementation uses an extensible Linux kernel module to measure the energy and latency implications of a system's processor without any manual intervention or external equipment. We demonstrate the practical applicability of our approach by analyzing two Intel processors from the Haswell and Skylake generation — an Intel Core i7-4790 and an Intel Core i7-6700K, respectively. The results show that our implementation yields reliable, precise, and reproducible measurements for the energy and latency implications of each processor's various idle states.

CCS CONCEPTS

• **Hardware** → *Chip-level power issues*; • **Software and its engineering** → *Software libraries and repositories*; • **Applied computing** → *Data centers.*

KEYWORDS

energy, measurement, idle states, wake-up latency, Intel processor, Haswell, Skylake, RAPL, HPET

*Equal contribution from the authors. Order randomized.

BDCAT '23, December 4–7, 2023, Taormina (Messina), Italy
© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM. This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive Version of Record was published in *IEEE/ACM 8th International Conference on Big Data Computing, Applications and Technologies (BDCAT '23), December 4–7, 2023, Taormina (Messina), Italy*, https://doi.org/10.1145/3632366.3632385.

ACM Reference Format:
Till Smejkal, Jan Bierbaum, Thomas Oberhauser, Horst Schirmeier, and Hermann Härtig. 2023. Sleep Well: Pragmatic Analysis of the Idle States of Intel Processors. In *IEEE/ACM 8th International Conference on Big Data Computing, Applications and Technologies (BDCAT '23), December 4–7, 2023, Taormina (Messina), Italy.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3632366.3632385

1 INTRODUCTION

Cloud data centers have seen remarkable growth in recent years, propelled by the surging demand for ubiquitous computing and storage capacity. With rising energy prices and increasing environmental concerns, the energy consumption of these facilities becomes a dominant aspect of their design and operation. Energy consumption accounts for up to 10 % to 15 % of a data center's total cost of ownership [2, 10, 18]. Every Joule of energy the computing hardware consumes requires matching cooling infrastructure for dissipating the resulting waste heat, further increasing the consumption. Koot and Wijnhoven "predict a combined growth of data center electricity needs of 286 TWh in 2016 until about 321 TWh in 2030, if all currently known growth factors remain the same" [19].

In typical cloud data centers, over-provisioning of resources, especially processors, is common: Operators strive to guarantee service level agreements (SLAs) [11], whereas for customers, misconfigurations can lead to over-provisioning [3]. In both cases, this practice leads to underutilization. Specialized low-power hardware states, known as *idle states*, reduce power consumption in this situation. We concentrate on x86 processors. These processors are widely employed in cloud environments and offer multiple idle states, each with different power consumption and wake-up latency characteristics. Selecting the most suitable idle state promises compelling energy savings during idle periods. However, high idle states[1] in processors can introduce high wake-up latencies, potentially impacting system performance and violating SLAs. As an example, our analysis of a Skylake Intel Core i7-6700K processor shows that in its highest idle state, the processor consumes as little as 2.7 % of the power its normal operation mode requires. On the downside, the processor will take ≈340 µs to resume normal execution once it entered this high idle state. To complicate matters further, shared resources, like caches, make the idle states of different cores interdependent.

[1] Sometimes also called deep sleep states.

Selecting the most suitable idle state for a particular core is, thus, a complex decision that trades wake-up latency off against energy savings and requires a global view of the system. Cloud systems typically rely on sophisticated algorithms known as *idle governors* for this purpose. These algorithms require precise information for their decision-making, whereas official documentation offers only information on the effects of idle states: The Intel manual, for example, provides coarse descriptions of the measures taken for each idle state yet lacks any concrete information regarding wake-up latency or energy consumption. Without this crucial data, idle governors are bound to select sub-optimal idle states, leading to inefficiencies in overall energy usage [12].

This paper introduces our novel approach to analyzing the energy and latency characteristics of idle states in modern Intel processors. We implement an open-source measurement framework as a Linux kernel module that takes full control of the system's processor and thereby excludes any disturbance by other software, including the kernel. Once in control, our framework traverses the processor's idle states and measures each state's effect on energy consumption ("How much power will I save when transitioning the processors to this idle state?") and wake-up latency ("When the processor resides in this idle state, how long will it take to start executing instructions again?").

Due to the well-controlled environment, the measurement results are stable, and we require only few repetitions, which results in fast measurements (<20 seconds). Employing RAPL and processor-internal timers for its measurements, our implementation is fully self-contained and can run autonomously. All in all, our implementation enables precise measurements of the equipped processor, thereby allowing the kernel's idle governor to make better decisions. These measurements run, for example, during the installation procedure of the operating system and, thus, incur no overhead in the running system.

To our knowledge, this approach is the first that (1) acquires as precise measurements as possible by tightly controlling the system, (2) requires only short measurement intervals, (3) is agnostic towards the underlying microarchitecture, and (4) performs all measurements without manual intervention or additional metering hardware. These properties are essential for our long-term goal of integrating the framework into the setup procedure of the operating system to provide the idle governor with a solid foundation for its decisions. Providing precise idle state characteristics can help to improve the system performance and energy consumption without any support and changes outside of the operating system kernel.

As a concrete demonstration, we measured two Intel processors from the Haswell and Skylake generation — an Intel Core i7-4790 and Core i7-6700K, respectively — which are still widely deployed in data centers. Our results show that, for the highest supported idle state, the Skylake offers energy saving potential superior to the Haswell (Skylake: 97.3 %; Haswell: 90.1 %) but also incur significantly longer wake-up latencies (Skylake: 340 µs; Haswell: 56 µs).

The remainder of the paper is structured as follows: Section 2 introduces the technical background of modern processors, including idle states and how to use them. In Section 3, we present our measurement infrastructure, followed by results for two different Intel architectures in Section 4. Section 5 discusses related work. Finally, Section 6 summarizes our work and outlines potential future extensions.

State	Effects
C0	Active
C1	Auto halt
C1E	Auto halt with lowest DVFS setting
C3	L1 & L2 cache flushed; Core clock gated
C6	Core power gated
C7	C6 + all cores in C7 → LLC flushed & power gated
C8	C7 + LLC flushed & power gated

Table 1: Idle states and their implementations as described in the vendor documentation for Haswell [15] and Skylake [14] Intel processors.

2 ENERGY-RELATED PROCESSOR FEATURES

Modern x86 processors offer multiple energy-related mechanisms; in the context of this paper, we concentrate on Dynamic Voltage and Frequency Scaling (DVFS), low-power operating states (idle states), and Running Average Power Limit (RAPL).

2.1 DVFS

Higher frequencies allow the processor to execute instructions faster. On the downside, higher frequencies necessitate higher supply voltages for stable operation. A processor's power is approximately proportional to its frequency f and the square of its voltage V [22, Ch. 2]: $P \approx ACfV^2$, where the capacitance C is a design-time constant and the activity A can be changed only by shutting off the clock when the processor idles; see Section 2.2. Today's processors allow adjustment of their frequency at runtime and automatically adapt their voltage accordingly. This feature permits the reduction of frequency (and hence voltage) during periods of low demand, thereby conserving energy. Linux uses so-called *scaling governors* to control this processor feature [24].

2.2 Idle States

When there is no work for the processor, it can enter dedicated low-power *idle states* to save energy. The Advanced Configuration and Power Interface (ACPI) defines a set of power states for hardware. Processor power states are named C0, C1, …, C*n* [29, Ch. 8]. ACPI only mandates support for C0 and C1, but contemporary processors offer many additional idle states. Note that not all intermediate idle states necessarily exist. For example, the two Intel processors we evaluate in Section 4 support idle states up to C7 (Core i7-4790) and C8 (Core i7-6700K) but "lack" states C2, C4, and C5.

Whereas in C0, the processor is fully operational and actively executes instructions, higher integers indicate idle states with increasingly aggressive energy saving measures. The concrete actions taken are vendor- and architecture-specific. They range from just halting execution in low idle states to completely shutting off the clock (*clock gating*) and voltage (*power gating*) for parts of the processor in higher idle states [22, Ch. 2].

These energy-saving measures come at a price, though: Transitioning from the active state C0 to an idle state Cx and back requires effort and time, e.g., for de/repowering circuitry and flushing/repopulating caches. Consequently, increasing values of x promise higher energy savings, but also incur higher entry and exit latencies [17, Vol. 3, Sec. 15.7]; also see Section 4.2. Processors offer special hardware instructions to transition to idle states. On contemporary Intel processors, the command pair `monitor`/`mwait` serves this purpose best; more details are in Section 2.3. In the Linux kernel, the *idle governor* is responsible for selecting the most suitable idle state for each core [30].

For years, processors have been featuring multiple cores and often also multiple hardware threads or sibling cores — Simultaneous Multi-Threading (SMT) or "Hyperthreading" in Intel terminology[2]. Internally, processors implement idle states for each individual core (*CC-states*) as well as for the processor as a whole (*PC-states*). Only the former can be requested explicitly; Section 2.3 will provide further details. In the case of SMT, a core's C-state is limited by the lowest C-state of any of its sibling cores and, similarly, the PC-state cannot be higher than the lowest CC-state of that processor, i.e. $PC \leq \min_{i \in \text{cores}} CC_i$. Other factors can also hinder the processor from reaching high C-states, and Intel offers no guarantees whatsoever about entering a desired idle state. The two Intel processors we evaluate in Section 4 support idle states up to C7 (Core i7-4790) and C8 (Core i7-6700K), respectively. Table 1 shows these idle states and the energy-saving measures taken in each, according to vendor documentation.

2.3 `monitor/mwait`

On state-of-the-art Intel processors, the most flexible way to request idle states is the `monitor`/`mwait` pair of instructions, both of which are only available in the privileged mode of the processor [17, Vol. 2, Sec. 4.3]. Figure 1 demonstrates how to use `monitor`/`mwait` for transitioning to idle state Cx: `monitor` "sets up an address range for the monitor hardware" ❶, i.e., the instruction specifies a memory address the processor should monitor for changes. `mwait` then requests ❷ the processor to pause execution and enter a specific idle state Cx until (1) the monitored memory location is modified or (2) an interrupt is triggered ❸.

According to vendor documentation, "[i]mplementation-specific conditions may result in an interrupt causing the processor to exit the implementation-dependent-optimized state even if interrupts are masked [...]." In general, there are no strict guarantees about which idle state the processor will actually enter or for how long it will remain there. Our measurements (Section 4) indicate that the processor, indeed, remains in the requested idle state most of the time.

2.4 Energy Measurement via RAPL

Whereas the intended purpose of the Running Average Power Limit (RAPL) is to allow the user to set hardware-enforced power limits, it also measures the energy consumption of Intel processors. Special model-specific registers (MSRs) allow software access to

[2]We use the term *core* to refer to all hardware threads of the processor. Cores sharing the same execution engine, we call sibling cores.

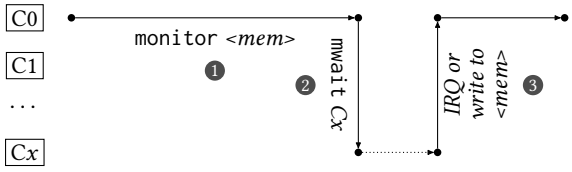


Figure 1: Using `monitor`/`mwait` to enter idle state Cx.

these measurements, which Hackenberg et al. verified to be highly accurate for processor generations starting with Haswell [8].

An important constraint of RAPL is its limited temporal and spatial granularity: Specifically, it cannot measure energy consumption on a per-core basis, only for the processor as a whole, and the energy counters update at discrete intervals of approximately one millisecond [17, Vol. 3B, Sec. 15.10.4]. Due to its ubiquitous availability, ease of use, and high precision, RAPL has seen wide adoption nevertheless. The research community has addressed the aforementioned limitations so that RAPL is suitable for measurements of VMs [4], individual processes [28], and short code paths [9].

3 METHODOLOGY

Accurately measuring the energy consumption of the individual idle states of an x86 processor comes with various challenges. This section describes our measurement approach in detail, points out the main challenges, and outlines how we overcame them. We designed our framework to (1) acquire as precise measurements as possible, (2) be generic in terms of applicability to different microarchitectures, and (3) require no manual intervention during the measurement but run fully automated.

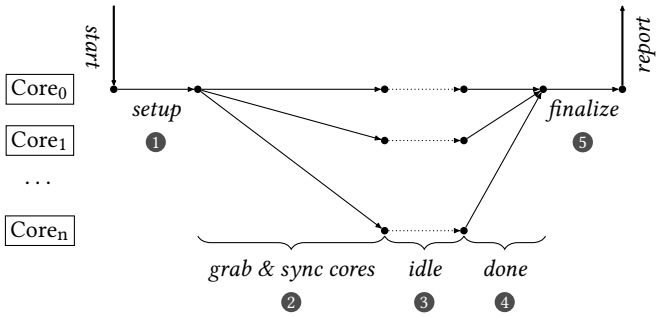


Figure 2: The general work flow of the measurement algorithm.

Figure 2 outlines the general structure of our measurement algorithm. After the user initiates a measurement, the algorithm performs some setup steps, such as initializing and activating internal counters and preparing the system for the measurement ❶. When this step is complete, the algorithm takes over all available cores in the system to create a fully controlled environment ❷. Next, the algorithm synchronizes all cores to ensure that the measurements begin simultaneously on all of them. The measurement itself is a controlled idle loop (see Section 3.2) on all cores in parallel ❸. At a user-specified point in time, the algorithm stops the idle loop, saves various performance counters, and releases all cores to the system

```
void setup(bool leader) {
  if (leader) {
    prepare_measurements();
    activate_perf_counter();
    run_on_cores(all & ~THIS_CORE, setup, false); (1)
  }

  if (leader)
    mask_interrupts(all & ~HPET); (2)
  else
    mask_interrupts(all & ~IPI);

  idle_loop(leader, IDLE_STATE, IDLE_TIME_MS);
}
```

Listing 1: Pseudo-Code of the setup procedure.

```
void idle_loop(bool leader,
               int idle_state,
               int idle_time_ms) {
  synchronize(); (3)
  if (leader)
    programm_timer(current_time() + idle_time_ms); (4)

  monitor(&dummy); (5)
  mwait(idle_state);

  if (leader)
    wakeup_followers();
}
```

Listing 2: Pseudo-Code of the idle measurement routine.

again (4). Finally, the algorithm calculates the corresponding energy consumption of the measuring step, determines wake-up latencies for the selected idle state, and makes the collected data available to the user (5).

During the design of our measurement framework, we already envisioned future extensions to other combinations of hardware and software configurations. Hence, in the following, we will first explain the general approach of the algorithm, followed by a description of the details we use for our implementation on the evaluated Intel x86 systems (see Section 4).

3.1 Controlling the Environment

The biggest challenge we needed to overcome for our approach was that accurate idle measurements require a closely controlled environment. As described in Section 2.2, a core will enter and remain in an idle state only if it does not execute any code and no interrupts or similar events occur. Hence, we decided to implement our measurement algorithm as an operating-system component. Running the algorithm within the operating-system kernel provides the following advantages: (1) No user-space applications will interfere with the measurement as we control the scheduler. (2) We have direct access to the hardware, e.g., for managing interrupts or reading performance counters, and (3) it is possible for the algorithm to use the plain `monitor` and `mwait` instructions to enter the processor's idle states.

This decision limits our approach to users who possess root access to the system being measured, but we argue that this is not a problem in general. Typically, it is not ordinary users who take such measurements but rather system administrators with the necessary rights to install or enable additional operating-system components, such as our idle-state measurement tool.

However, just entering the operating-system kernel will only give our implementation exclusive control over one core in the system (in the following called the *leader core*). In order to ensure accurate measurements of the energy and latency properties of idle states, it is imperative to maintain complete control over the entire system. Hence, we need to perform additional steps as outlined in Listing 1. We use an interface that allows kernel components to run arbitrary functions in parallel on all available cores, thus granting us full control over the system (1). By invoking this interface on the leader core, it sends an inter-processor interrupt (IPI) to all other cores (the *follower cores*), which stops their current execution and forces them to execute the specified function.

Unrelated events at external components, such as the network adapter or the keyboard, could interfere with the controlled idling of cores (see Section 2.3). We avoid this issue by masking all non-essential interrupts (2). In order to wake up after the idle period, we need to allow specific interrupts (see Section 3.2 for more details).

Measurements with Linux. We implemented the outlined approach in a Linux kernel module. While a module allows for deep integration into the Linux kernel, users can dynamically add it to or remove it from an existing kernel at runtime. This feature simplifies the use of our framework. To gain full control of the system, we utilize the `on_each_cpu`[3] interface of the Linux kernel. This function provides exactly the aforementioned properties, i.e., it allows us to run arbitrary code on all cores of the system and, thereby, actively manage the system during our measurements.

3.2 Controlling the Idle and Wake Up

Once the system is prepared, the actual idle routine outlined in Listing 2 starts. First, all cores use a barrier to synchronize again so the idle phase will start perfectly in parallel (3). Afterward, the cores execute the `monitor`/`mwait` instruction sequence (5). This instruction sequence stops execution on each core and requests the cores to enter the designated idle state (see Section 2.3).

Since entering an idle state with masked interrupts will block the core forever, we need a way to wake up from the idle phase in a controlled fashion. The leader core takes care of this by setting a wake-up call that triggers after a pre-specified period of time (4). There are two options to achieve this wake-up: We can either set a timer and allow the associated interrupts, or we need an external device (or a processor on another socket) to write to the monitored memory region. After the leader core wakes, the follower cores need to leave their idle states as well. Again, we can either write to the memory region the follower cores monitor or send them an enabled interrupt.

[3] https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/tree/include/linux/smp.h?h=v6.2.9#n69

Waking Up with HPET and IPIs. In our implementation, we opted for the high-precision event timer (HPET) to wake the leader core. This timer offers a frequency of at least 10 MHz (100 ns tick period) and a drift rate of less than ±0.05 % over any interval longer than 1 ms [16]. Initialized appropriately, the HPET interrupts the leader core once the idle period is over. This technique allows for very fine-grained idle periods and is guaranteed to work in every circumstance. To wake up the follower cores, we use inter-processor interrupts (IPIs). In both cases, we ensure that the respective interrupts — HPET for the leader core and IPI for the follower cores — are not masked during setup ❷.

Unfortunately, using the HPET to wake up the leader core currently requires a few changes to the Linux kernel. By default, Linux does not grant kernel modules direct access to the HPET. We added or changed about 50 lines of code in the kernel to allow this fine-grained control. In the future, we plan to replace the HPET with a different interrupt source that is, without any kernel modifications, directly accessible from kernel modules.

3.3 Accurate Measurements

A key goal of our framework is to acquire as precise measurements of idle states as possible. With the ability to control the environment in a way that allows all cores to enter and maintain specific idle states, we have met all necessary prerequisites for precise measurements. Our framework focuses on measuring the energy consumption and the wake-up latency of the different idle states available on the system under test.

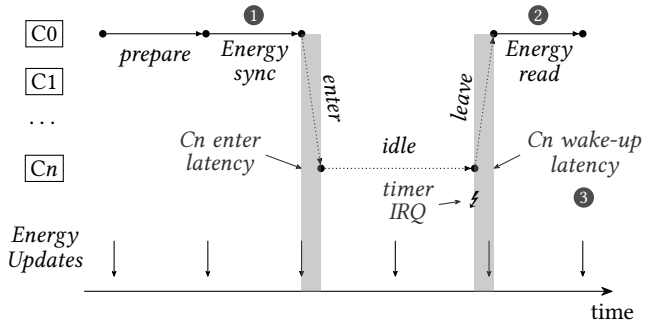


Figure 3: Detailed workflow of the energy and wake-up latency measurements of the idle states.

Figure 3 outlines the workflow we follow to gather the data of interest. For fine-grained measurements, we apply a technique similar to the one described by Hähnel et al. [9]: We synchronize with the energy measurement updates before entering the idle states ❶ to always start with an up-to-date energy value. This approach is especially important for energy measurement systems with low update rates because, otherwise, short idle times would lead to significant measurement errors. After synchronizing with the energy counter update, we save its value as a basis, program the wake-up timer (see Section 3.2), and finally enter the selected idle state. After the wake-up, we optionally synchronize again with the energy measurements ❷. Using these two values, we can precisely calculate the consumed energy. When properly accounting for the synchronization phases with the measurement, this approach is viable even for short idle periods, as Hähnel et al. showed.

To determine the wake-up latency of the processor's idle states, we use high-precision time stamps. Since we program the point in time when the timer triggers the interrupt to wake up the leader core, we know the exact time the core leaves the idle state. Once the leader core enters the C0 active state, our framework immediately takes another time stamp ❸. With this time stamp, we can calculate the wake-up latency as the difference between the configured time stamp of the timer and the one measured in the C0-state.

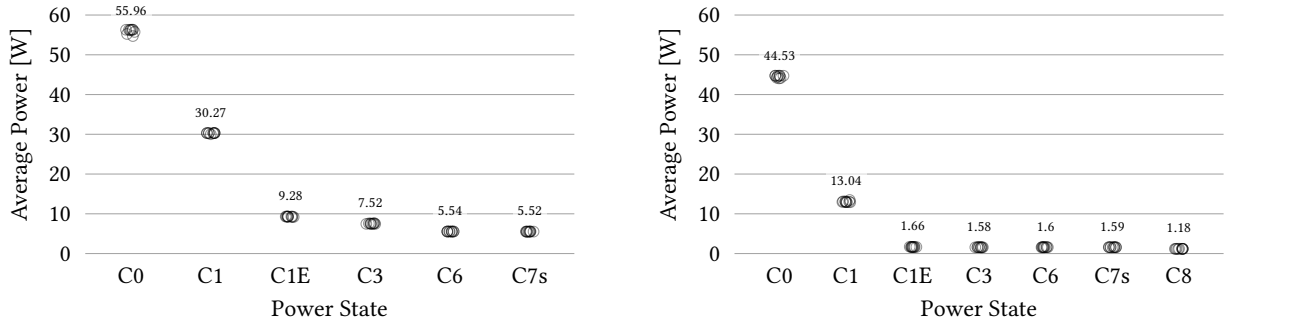
Measurements with RAPL and HPET. Modern Intel processors come with an energy measurement mechanism on board. In our implementation, we use the RAPL energy counters. As described in Section 2.4, these counters are accessible through model-specific registers (MSRs) of the processor, are very precise, and update approximately every millisecond. This combination of properties allows fine-grained energy measurements and makes RAPL perfect for our measurement algorithm. For sleep times longer than 50 milliseconds, we omit the synchronization with the energy counters after the idle phase. The additional measurement error introduced by directly reading the RAPL counters is negligible in these cases [9].

We rely on the HPET for the latency measurement. Immediately after entering the C0-state on the leader core, we read the current HPET counter and use this value as a time stamp. By comparing the read value with the programmed counter value at which the HPET triggered the interrupt, we can calculate the wake-up latency of the idle state that the core used during `monitor`/`mwait`.

4 RESULTS

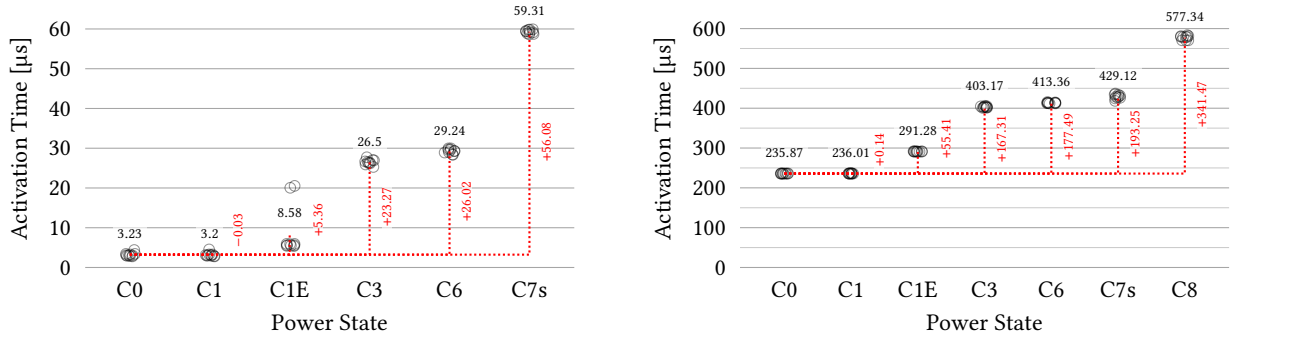
To demonstrate the applicability of our measurement infrastructure, we analyzed various idle-state-related aspects of two Intel processors from the Haswell [15] and Skylake [14] generations: an Intel Core i7-4790 and an Intel Core i7-6700K, respectively. Both processors feature eight cores, four physical ones with two hardware threads each. The newer Core i7-6700K supports power states up to idle state C8, whereas the Core i7-4790 stops at C7s. According to the internal capability register [16], the HPET tick periods are ≈ 42 ns on the Core i7-6700K and ≈ 70 ns on the Core i7-4790. To avoid any side effects of DVFS (see Section 2.1) and best show the energy savings achievable by their idle states, we run the processors at their maximum regular frequency: 3.60 GHz for the Core i7-4790 and 4 GHz for the Core i7-6700K. As our work is motivated by providing an accurate database for the system's idle governor to make informed decisions, we restrict our experiments to the processor power states used by Linux's idle governor.

We repeat each measurement ten times to identify potential stability problems of the measured values. The idle phase lasts 100 ms for all experiments. This way, the processor has enough time to prepare and enter the requested power state, yet the measurements are short enough to reliably analyse a processor in less than 20 seconds. 100 ms is also long enough to omit synchronizing with the RAPL counters after returning to the active state C0 (see Section 3.3). When using `mwait`, we can only explicitly request a core's power state (refer to Section 2.2). Therefore, we use the term "C-state" or



(a) Haswell Core i7-4790 **(b) Skylake Core i7-6700K**

Figure 4: Processor power by requested power state. Individual measurements are depicted as circles with a horizontal jitter to avoid full overlap of identical values. Numbers next to the data points indicate mean values.



(a) Haswell Core i7-4790 **(b) Skylake Core i7-6700K**

Figure 5: Time between the programmed HPET interrupt and the HPET counter taken when resuming execution (see Section 3.3). Individual measurements are depicted as circles with a horizontal jitter to avoid full overlap of identical values. Numbers next to the data points indicate mean values. The additional time required for idle states is the state's corresponding wake-up latency (shown as red numbers).

C*x* instead of the more accurate CC*x* when it is unnecessary to distinguish it from processor-wide PC power states.

Unfortunately, to the best of our knowledge, there is little research that targets precise measurements of the characteristics of individual idle states (see Section 5) and no openly available implementations. Hence, we cannot directly compare against existing solutions.

4.1 Energy Consumption/Power

First, we look into the effects of the processors' power states on their energy consumption. In this experiment, all cores of the processor call `mwait` to transition to a specific power state. Note that this is also true for state C0. To enhance comprehension, Figure 4 shows the average power of the processor during the measurement, i.e., the energy reported by RAPL divided by the 100 ms duration of the idle phase. Overall, the newer Core i7-6700K (Figure 4b) consumes less power than the Core i7-4790 (Figure 4a) in all power states, although the Core i7-6700K has a higher TDP.

For both processors, we observe a particularly steep decline in power for the first two idle states, C1 and C1E; dropping from 56 W to 30 W (≈ 54 % of C0's power) to 9 W (≈ 16 %) for the Core i7-4790 and from 45 W to 13 W (≈ 29 %) to 1.7 W (≈ 4 %) for the Core i7-6700K. Even though higher idle states do further reduce the processors' power, the effect is much less pronounced. The lowest values we observed are 5.5 W (≈ 10 %) for the Core i7-4790 in C7s and 1.2 W (≈ 2.5 %) for the Core i7-6700K in C8. By repeating the same experiment on the lowest DVFS setting, we verified that C1E indeed only reduces the processor's frequency to its minimum. When selecting this minimum frequency manually, states C1 and C1E become indistinguishable with regard to power.

During our work, we discovered cores waking up before the intended idle period was over. Whereas it is in line with the vendor documentation [17, Vol. 2, Sec. 4.3] that "[i]mplementation-specific conditions [...] result in an interrupt causing the processor to exit the implementation-dependent-optimized state even if interrupts are masked [...]", such behavior would distort our measurements if left unattended. To counteract the impact, we reenter the idle state via `mwait` in cases where it returns prematurely. This issue occurs infrequently (less than five times per measurement) and only for idle states beyond C1/C1E. The vendor documentation

provides a possible explanation, stating that "A System Management Interrupt (SMI) handler returns execution to either Normal state or the C1/C1E state." [14, 15, Sec. 4.2.4].

When requesting `mwait` to use the active power state C0, the command returns immediately. Thus, for C0, our framework effectively calls `mwait` in a tight loop, and we see 640 000 to 700 000 wake-ups in the 100 ms measurement period.

4.2 Wake-Up Latency

Returning a core from an idle state Cx to its active state C0 takes time. The duration of this process depends on x, as higher idle states activate more complex energy-saving mechanisms that the processor must reverse before it can resume its normal operation. Figure 5 shows the time the processor takes to resume normal execution when the HPET interrupt ends the idle phase in different power states. For C0, this time is the overhead incurred by the interrupt itself, which we consider the baseline. The extra time taken in idle states is the state's respective wake-up latency. We calculate this wake-up latency as described in Section 3.3 by taking precise time stamps directly when the leader core enters the C0 active state. All time stamps are based on the HPET's counter. By calculating the difference between the wake-up time stamp and the configured wake-up timer interrupt, we can precisely determine the actual latency.

Again, there is a clear difference between the two processors. The Skylake Core i7-6700K starts with a much higher baseline (235 µs vs. 3 µs for the Haswell Core i7-4790) and, in general, exhibits higher latencies. A thorough analysis of the cause is beyond the scope of this paper, but we presume that changes in the Skylake microarchitecture adversely affect the involved state transitions. Due to its strong isolation properties, "Software Guard Extensions" (SGX) [5], a feature set originally introduced with Skylake, are a likely source. The wake-up latencies for C1 and C1E are moderate (< 5.5 µs for the Core i7-4790 and < 56 µs for the Core i7-6700K), but higher idle states increase these values to up to 56 µs and 342 µs, respectively.

These findings reflect our observations of the idle states' effects on power. The same is true for states C1 and C1E becoming indistinguishable, this time regarding wake-up latency, when the processor already runs at its minimum frequency.

Although the wake-up latencies, and especially the differences between the two processor generations, are surprising, the order of magnitude of our measured latencies for the different idle states is in line with other research on this topic [20, 31].

4.3 Number of Idle Cores

For SMT-capable processors like those in our setup, a core can only enter a specific idle state when its sibling core joins it (see Section 2.2). We were able to verify this principle by transitioning increasing numbers of cores to an arbitrary idle state, C1 in the example depicted in Figure 6. Active, non-idle cores run a tight busy loop in this experiment to ensure the hardware does not transition them to an idle state. When increasing the number of idle cores, we first add sibling cores. So, for every even number of idle cores, the corresponding physical cores may actually enter state C1.

The graph also illustrates how the power consumption decreases when additional core pairs enter the idle state: For the Haswell Core i7-4790, each pair in C1 results in an approximate reduction of 8 W, whereas for the Skylake Core i7-6700K it is about 9 W. However, the power drop is different for both processors when all the cores become idle. The Core i7-4790 experiences a power drop similar to other core pairs, whereas the Core i7-6700K has a higher power drop of approximately 11 W. The reason is likely a more efficient implementation of the processor-wide idle state PC1 in the Skylake, which becomes available once all cores enter CC1.

Note the slight difference between zero cores idling in Figure 6 and all cores being in state C0 in Figure 4. One may assume that in both scenarios, the full processor is in state C0 executing instructions, thus resulting in equivalent power usage. In the given situation, the distinction lies in the fact that for state C0 in Figure 4, all cores continuously execute `mwait`. On the other hand, in Figure 6, when there are no idle cores, all cores run busy loops. The research community is well aware that different instructions have distinct effects on the processor's energy consumption [28].

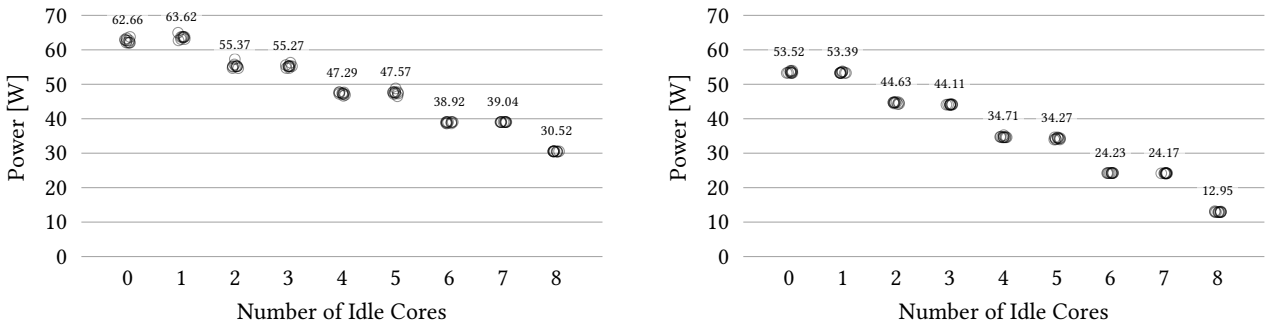
4.4 Internal Idle States

Even though `mwait` allows software to request a core's power state (CC-state), there are no guarantees that this state will actually become active. In addition, processor-wide power states (PC-states) are utterly beyond user control. Intel processors offer model-specific registers (MSRs), however, that allow us to monitor for how long individual cores and the processor as a whole stayed in specific power states [17, Vol. 4]. Using these MSRs, we can verify that for both analyzed processors the individual cores indeed enter the requested CC-states. Figures 7a and 7b show the fraction of time each core remains in a particular CC-state after requesting a specific C-state for all the processor's cores. For simplicity and due to the absence of significant variation between cores, we show values aggregated across all cores. This aggregation may cause small rounding errors, however.

The available MSRs do not cover all idle states; neither CC1/CC1E nor PC0 have corresponding counters. We, therefore, measure the actual length of the idle interval and list the time unaccounted for by the other power states as CC1/CC1E in Figures 7a and 7b and as PC0 in Figures 7c and 7d. Minor inaccuracies may arise due to the different time sources involved in this approach. This shortcoming does not apply to the main contribution of our work, namely the accurate measurement of the energy implications and wake-up latency of idle states.

In contrast to the cores, the processor as a whole does not enter all its idle states. Our observations are in line with the official documentation for idle states up to C3: The processor will only enter PC3 once all cores are at least in CC3 [14, 15, Sec. 4.2.5]. Even with all cores residing in higher CC-states, however, we do not observe PC states beyond PC2. A thorough analysis of the cause is out of the scope of this paper, but likely components besides the cores keep the processor from entering higher idle states.

Neither processor remains in higher idle states throughout the entire idle period, and the Skylake Core i7-6700K is particularly affected by this. This observation applies to both the CC-states and the PC-states. One likely explanation is the execution of `mwait` itself. We can only collect reference time stamps and MSR values before running `mwait`, repeatedly if it returns prematurely (Section 4.1), so



(a) Haswell Core i7-4790 **(b) Skylake Core i7-6700K**

Figure 6: Processor power with a specific number of cores in idle state C1. Only when both sibling cores enter an idle state the associated energy saving measure become active. Individual measurements are depicted as circles with a horizontal jitter to avoid full overlap of identical values. Numbers next to the data points indicate mean values.

	C0	C1	C1E	C3	C6	C7s
CC7	0	0	0	0	0	99.92
CC6	0	0	0	0	99.92	0
CC3	0	0	0	99.93	0	0
CC1/CC1E	0	99.99	99.98	0.05	0.06	0.06
CC0	100	0.01	0.02	0.02	0.02	0.02

(a) Haswell Core i7-4790 — Cores (aggregated)

	C0	C1	C1E	C3	C6	C7s	C8
CC7	0	0	0	0	0	99.09	99.11
CC6	0	0	0	0	99.09	0	0
CC3	0	0	0	99.11	0	0	0
CC1/CC1E	0.43	99.95	99.94	0.84	0.86	0.86	0.84
CC0	99.57	0.05	0.06	0.05	0.05	0.05	0.05

(b) Skylake Core i7-6700K — Cores (aggregated)

	C0	C1	C1E	C3	C6	C7s
PC7	0	0	0	0	0	0
PC6	0	0	0	0	0	0
PC3	0	0	0	0	0	0
PC2	0	0	0	99.88	99.86	99.83
PC0	100	100	100	0.12	0.14	0.17

(c) Haswell Core i7-4790 — Whole Processor

	C0	C1	C1E	C3	C6	C7s	C8
PC7	0	0	0	0	0	0	0
PC6	0	0	0	0	0	0	0
PC3	0	0	0	0	0	0	0
PC2	0	0	0	98.9	98.86	98.86	98.97
PC0	100	100	100	1.1	1.14	1.14	1.03

(d) Skylake Core i7-6700K — Whole Processor

Figure 7: Distribution of internal idle states for the whole processor (PCx) and its cores (CCx) by requested idle state. The same idle state was requested for all cores. Darker shades of color correspond to more time spent in a particular state. The number inside each field gives the percentage of time the processor remained in the respective state. Due to the measurement approach and aggregation over individual cores, there may be small inaccuracies in the results.

the processor remains in C0 for this short period of time. Another option is that the processor "progressively" transitions to higher idle states when additional power-saving mechanisms activate.

4.5 Generalizability of the Framework

Due to space constraints, this paper only discusses the results of two specific Intel systems — a Haswell Core i7-4790 and a Skylake Core i7-6700K. We also ran measurements on other processors to verify the generalizability of our framework. For example, we also applied our framework to an Intel Haswell Core i5-4300U. This system is designed for mobile use, has a low thermal design power of 15 W and supports idle states up to C10.

The results from this Core i5-4300U show similar trends as those we present for the Core i7-4790: Deeper idle states decrease the processor's average power consumption from ≈ 11.7 W in C0 to ≈ 2.9 W (≈ 24.8 % of C0's power) starting from C7s. The corresponding wake-up latencies increase from ≈ 0.5 µs for C1 and ≈ 2 µs for C1E to ≈ 64.6 µs for C10. Like the results presented in this paper, the measurements of the Haswell Core i5-4300U are very stable. Even though both processors feature the Haswell microarchitecture, their characteristics vary significantly. This example shows that coarse heuristics are insufficient for the idle governor to make optimal decision. Our framework can provide reliable and precise data for the very processor installed in a given system.

5 RELATED WORK

The subject of x86 idle states and how to use them to save energy has been a heavily discussed research topic for over a decade. We differentiate the existing research into two main directions: (1) analysis of the properties of hardware performance states and (2) applying idle states to save energy during periods of minor system load.

Analysing Properties of Idle and Performance States. Hackenberg, Schöne, Ilsche, et al. have provided detailed analyses of the wake-up latencies and power consumptions of idle states available in modern x86 Intel processors [8, 13, 25, 27] as well as AMD processors of the Zen 2 generation [26]. Their method relies on user-space measurements and statistical methods, however, which increases the required measurement times and limits achievable accuracy. In contrast, our measurement framework tightly controls the environment to prevent noise from other hardware and software components. This setup allows us to acquire as precise measurements as possible.

A similar approach for a slightly different target was taken by Mazouz et al. [21]. They analyze the switching latency between different processor frequencies (DVFS performance states) of various Intel microarchitectures. Like Hackenberg et al., they base their work on statistical analysis of user space measurements, which limits their resilience to noise.

An analysis of the effect of idling outside the realm of powerful x86 Intel processors, which our work tackles, was done by Daud et al. [6]. In their work, they examine the energy consumption of embedded processors when idling and under load. They conclude that proper idle management in embedded devices can save significant amounts of energy and improve the overall usability of the devices due to the usually limited battery capacity.

Compared to the existing related work, our approach is a significant step forward because we are able to obtain very precise and reliable properties of idle states in a few seconds without manual intervention.

Using Idle States for Saving Energy. With modern processors, it has become common to use idle states to reduce the overall system energy consumption during idle periods. Various research groups tackled the problem of how to balance the relatively high wake-up latencies of idle states with the tight latency requirements for the completion of requests common in cloud environments. Examples include the work of Yahya et al. [31] and Antoniou et al. [1], who both envision a different idle-state hierarchy that is more compatible with the completion-latency requirements in the cloud. In their work, they propose to extend the typical idle states of x86 processors with additional ones that have better properties regarding power consumption and wake-up latency by using different hardware power-saving techniques.

Duan et al. take a different approach to achieve a similar goal [7]. They propose to integrate an idle-time prediction algorithm into the idle governor of the operating system and thereby select better-suited idle states. Instead of selecting as-high-as-possible idle states, they argue to use idle states whose wake-up latency matches the expected usage pattern of the system.

Changing the heuristics of the idle governor of the Linux kernel in order to use different idle states was also done by Ilsche et al. [12]. They identified problems in the default Linux idle governor that led to inefficiencies in the overall system energy consumption while idling. The idle governor mispredicted when the next system job will require an active processor core and, hence, selected a non-optimal idle state and wake-up timer. This combination led to significantly higher energy consumption while idling, thus wasting energy.

The work of Paya and Marinescu tries to tackle the problem of non-optimally-selected system idle states in cloud environments from another direction [23]. They consider the properties of the idle states available in the system already at the point in time when work gets distributed in the cloud. Instead of distributing work equally within the data center, they propose to leave servers explicitly idle so that they can enter higher idle states and thus save more energy.

All the discussed related work differs from our approach, as we focus on gathering a reliable ground truth of the properties of the processor's idle states. Our framework's measurements can improve those discussed in related work, as our results are more accurate and stable than existing approaches.

6 CONCLUSION

In this paper, we described our novel approach for precisely analyzing the energy consumption and wake-up latency of the idle states of contemporary Intel x86 processors. We implemented a framework[4] that can automatically evaluate the available idle states in a closely controlled environment, free of disturbance from system noise and shielded as good as possible from other software influences. To show the practicality of our implementation, we analyzed two different Intel processors from the Haswell and Skylake

[4] https://github.com/TUD-OS/SleepWell

generations — an Intel Core i7-4790 and an Intel Core i7-6700K, respectively. Our results demonstrate that our framework can measure the wake-up latency and power consumption of the idle states with very high precision and close to no variance. We could further verify the influence of sibling cores (SMT, Intel Hyper-Threading) on idle states and identify which processor-internal idle states (core CC-states and processor PC-states) the hardware transitions to.

Fully measuring a processor takes our framework less than 20 seconds, so it could be integrated into the installation procedure of the operating system to provide the idle governor with a solid foundation for its decisions. The idle governor would no longer have to rely on sub-optimal heuristics or incomplete information from an ACPI table to make decisions. Instead, it can use accurate measurements of the actual hardware.

Future Work. We plan to extend and generalize our framework to other x86 processor vendors (e.g., AMD) and even completely different architectures, such as Arm or RISC-V. As RAPL or similar energy measurement facilities are not necessarily available on other platforms, we want to enable the integration of external energy and power metering infrastructure. This extension will make our framework even more flexible but requires more complex handling and synchronization in the measurement routine. For direct practical application, we would like to completely embed our framework in an operating system kernel and use the results of the measurements to improve the idle governor and its idle-state selection algorithm.

Furthermore, we want to analyze the impact of the wake-up source on the characteristics of the idle states. To this end, we plan to complement the HPET interrupts used in this paper with techniques such as the local APIC timer interrupts and writes to the *monitored* memory region from a different processor socket.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their time and constructive comments. We are also grateful for our colleague Maksym Planeta's help in improving the paper's quality and presentation. This research was co-financed by the Federal Ministry of Education and Research of Germany in the program of "Souverän. Digital. Vernetzt." (joint project 6G-life, project ID 16KISK001K) and by public funding from the state of Saxony/Germany.

REFERENCES

[1] G. Antoniou, H. Volos, D. B. Bartolini, T. Rollet, Y. Sazeides, and J. H. Yahya. 2022. AgilePkgC: an agile system idle state architecture for energy proportional datacenter servers. (2022). DOI: 10.1109/MICRO56248.2022.00065.

[2] L. A. Barroso, U. Hölzle, and P. Ranganathan. 2019. *The Datacenter as a Computer: Designing Warehouse-scale Machines*. Springer Nature.

[3] J. Chapel. 2020. The cloud is booming — but so is cloud waste. (Mar. 4, 2020). Retrieved July 23, 2023 from https://devops.com/the-cloud-is-booming-but-so-is-cloud-waste/.

[4] M. Colmant, M. Kurpicz, P. Felber, L. Huertas, R. Rouvoy, and A. Sobe. 2015. Process-level power estimation in VM-based systems. In *Proceedings of the Tenth European Conference on Computer Systems* (EuroSys '15). European Conference on Computer Systems. ACM, Bordeaux, France, (Apr. 2015). ISBN: 978-1-4503-3238-5. DOI: 10.1145/2741948.2741971.

[5] V. Costan and S. Devadas. 2016. Intel SGX explained. (2016). https://eprint.iacr.org/2016/086.

[6] S. Daud, R. B. Ahmad, O. B. Lynn, Z. I. Abd Kareem, L. Munirah Kamarudin, P. Ehkan, M. N. M. Warip, and R. R. Othman. 2014. The effects of CPU load & idle state on embedded processor energy usage. In *2014 2nd International Conference on Electronic Design (ICED)*, 30–35. DOI: 10.1109/ICED.2014.7015766.

[7] L. Duan, D. Zhan, and J. Hohnerlein. 2015. Optimizing Cloud Data Center Energy Efficiency via Dynamic Prediction of CPU Idle Intervals. In *2015 IEEE 8th International Conference on Cloud Computing*, 985–988. DOI: 10.1109/CLOUD.2015.133.

[8] D. Hackenberg, R. Schöne, T. Ilsche, D. Molka, J. Schuchart, and R. Geyer. 2015. An energy efficiency feature survey of the Intel Haswell processor. In (IPDPSW '15). IEEE International Parallel and Distributed Processing Symposium Workshop. IEEE, Hyderabad, India, (May 2015), 896–904. ISBN: 978-1-4673-7684-6. DOI: 10.1109/IPDPSW.2015.70.

[9] M. Hähnel, B. Döbel, M. Völp, and H. Härtig. 2012. Measuring energy consumption for short code paths using RAPL. *ACM SIGMETRICS Performance Evaluation Review*, 40, 3, (Jan. 2012), 13–17. DOI: 10.1145/2425248.2425252.

[10] D. Hardy, M. Kleanthous, I. Sideris, A. G. Saidi, E. Ozer, and Y. Sazeides. 2013. An analytical framework for estimating TCO and exploring data center design space. In *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 54–63.

[11] C.-H. Hsu, Q. Deng, J. Mars, and L. Tang. 2018. Smoothoperator: reducing power fragmentation and improving power utilization in large-scale datacenters. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, 535–548.

[12] T. Ilsche, M. Hähnel, R. Schöne, M. Bielert, and D. Hackenberg. 2017. Powernightmares: the challenge of efficiently using sleep states on multi-core systems. In *Proceedings of the Workshop on Runtime and Operating Systems for the Many-Core Era* (ROME '17). Workshop on Runtime and Operating Systems for the Many-Core Era. Springer, Santiago de Compostela, Spain, (Aug. 2017), 623–635. ISBN: 978-3-319-75177-1. DOI: 10.1007/978-3-319-75178-8.

[13] T. Ilsche, R. Schöne, P. Joram, M. Bielert, and A. Gocht. 2018. System monitoring with lo2s: power and runtime impact of C-state transitions. In (IPDPSW '18). IEEE International Parallel and Distributed Processing Symposium Workshops. IEEE, Vancouver, BC, Canada, (May 2018), 712–715. ISBN: 978-1-5386-5555-9. DOI: 10.1109/IPDPSW.2018.00114.

[14] Intel Corporation. 2022. *6th Generation Intel® Core™ Processor Family: Datasheet - Volume 1.* (Feb. 2022). 164 pp. Retrieved July 23, 2023 from https://www.intel.com/content/www/us/en/content-details/332687/6th-generation-intel-core-processor-family-datasheet-volume-1.html.

[15] Intel Corporation. 2015. *Desktop 4th Generation Intel® Core™ Processor Family, Desktop Intel® Pentium® Processor Family, and Desktop Intel® Celeron® Processor Family: Datasheet – Volume 1 of 2.* (Mar. 2015). 125 pp. Retrieved July 23, 2023 from https://cdrdv2.intel.com/v1/dl/getContent/328897?fileName=4th-gen-core-family-desktop-vol-1-datasheet.pdf.

[16] Intel Corporation. 2004. *IA-PC HPET (High Precision Event Timers) Specification.* (Oct. 2004). 33 pp. https://www.intel.com/content/dam/www/public/us/en/documents/technical-specifications/software-developers-hpet-spec-1-0a.pdf.

[17] Intel Corporation. 2022. *Intel 64 and IA-32 Architectures Software Developer's Manual.* (Dec. 2022). 5060 pp. https://software.intel.com/en-us/download/intel-64-and-ia-32-architectures-sdm-combined-volumes-1-2a-2b-2c-2d-3a-3b-3c-3d-and-4.

[18] J. Koomey, K. Brill, P. Turner, J. Stanley, and B. Taylor. 2007. A Simple Model for Determining True Total Cost of Ownership for Data Centers. *Uptime Institute White Paper, Version*, 2, 2007.

[19] M. Koot and F. Wijnhoven. 2021. Usage impact on data center electricity needs: a system dynamic forecasting model. *Applied Energy*, 291, 116798. DOI: 10.1016/j.apenergy.2021.116798.

[20] N. Kurd et al. 2015. Haswell: A Family of IA 22 nm Processors. *IEEE Journal of Solid-State Circuits*, 50, 1, 49–58. DOI: 10.1109/JSSC.2014.2368126.

[21] A. Mazouz, A. Laurent, B. Pradelle, and W. Jalby. 2014. Evaluation of CPU frequency transition latency. *Computer Science-Research and Development*, 29, 3-4, 187–195.

[22] P. R. Panda, B. V. N. Silpa, A. Shrivastava, and K. Gummidipudi. 2010. *Power-Efficient System Design*. Springer Science & Business Media.

[23] A. Paya and D. C. Marinescu. 2017. Energy-aware load balancing and application scaling for the cloud ecosystem. *IEEE Transactions on Cloud Computing*, 5, 1, 15–27. DOI: 10.1109/TCC.2015.2396059.

[24] Rafael J. Wysocki. 2017. CPU performance scaling — the Linux kernel documentation. (2017). https://www.kernel.org/doc/html/latest/admin-guide/pm/cpufreq.html.

[25] R. Schöne, T. Ilsche, M. Bielert, A. Gocht, and D. Hackenberg. 2019. Energy efficiency features of the Intel Skylake-SP processor and their impact on performance. In *International Conference on High Performance Computing & Simulation* (HPCS '19). IEEE, Dublin, Ireland, (July 2019), 399–406. ISBN: 978-1-72814-484-9. DOI: 10.1109/HPCS48598.2019.9188239.

[26] R. Schöne, T. Ilsche, M. Bielert, M. Velten, M. Schmidl, and D. Hackenberg. 2021. Energy Efficiency Aspects of the AMD Zen 2 Architecture. In *2021 IEEE International Conference on Cluster Computing (CLUSTER)*, 562–571. DOI: 10.1109/Cluster48925.2021.00087.

[27] R. Schöne, D. Molka, and M. Werner. 2015. Wake-up latencies for processor idle states on current x86 processors. *Computer Science - Research and Development*, 30, 2, (May 2015), 219–227. DOI: 10.1007/s00450-014-0270-z.

[28] T. Smejkal, M. Hähnel, T. Ilsche, M. Roitzsch, W. E. Nagel, and H. Härtig. 2017. E-Team: practical energy accounting for multi-core systems. In *Proceedings of the 2017 USENIX Conference on Usenix Annual Technical Conference* (USENIX ATC '17). USENIX Annual Technical Conference. USENIX Association, Santa Clara, CA, USA, (July 2017), 589–601. ISBN: 978-1-931971-38-6. https://www.usenix.org/conference/atc17/technical-sessions/presentation/smejkal.

[29] UEFI Forum, Inc. 2022. *Advanced Configuration and Power Interface (ACPI) Specification*. (Release 6.5 ed.). (Aug. 29, 2022). 1126 pp. https://uefi.org/sites/default/files/resources/ACPI_Spec_6_5_Aug29.pdf.

[30] R. J. Wysocki. 2018. CPU idle time management — the Linux kernel documentation. (2018). Retrieved July 23, 2023 from https://www.kernel.org/doc/html/latest/admin-guide/pm/cpuidle.html.

[31] J. H. Yahya et al. 2022. AgileWatts: an energy-efficient CPU core idle-state architecture for latency-sensitive server applications. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 835–850. DOI: 10.1109/MICRO56248.2022.00063.