Energy Efficiency Features of the Intel Alder Lake Architecture

Robert Schöne
Markus Velten
Daniel Hackenberg
Thomas Ilsche
robert.schoene@tu-dresden.de
markus.velten@tu-dresden.de
daniel.hackenberg@tu-dresden.de
thomas.ilsche@tu-dresden.de
CIDS, Information Services and High Performance Computing (ZIH)
TU Dresden
Dresden, Germany

ABSTRACT

Intel's first heterogeneous processor, Alder Lake, combines two different core architectures from the Core and Atom families: Golden Cove and Gracemont, respectively. While the heterogeneity of this chip can improve performance and energy efficiency, it also increases the complexity of scheduling decisions and power saving mechanisms. In this paper, we analyze performance and energy characteristics of an Alder Lake system and describe effects of power saving mechanisms. We evaluate the factors that influence the time required to switch core and uncore frequencies and waking cores from idle states. In addition, we assess the efficiency of the two core architectures across various workloads. We show that in states with low power consumption, RAPL energy measurements are inaccurate, and actual (externally measured) power consumption also exhibits peculiar patterns. Through experiments, we also examine the newly introduced user space idle states, and the novel telemetry capability. This information can be used by other researchers to design efficient software and further experiments, and explain measured performance on heterogeneous Intel processors.

CCS CONCEPTS

• **Computer systems organization** → **Multicore architectures**; **Heterogeneous (hybrid) systems**; • **Hardware** → **Chip-level power issues**; **Platform power issues**.

KEYWORDS

Intel, power management, energy efficiency, DVFS, idle state, RAPL

ACM Reference Format:
Robert Schöne, Markus Velten, Daniel Hackenberg, and Thomas Ilsche. 2024. Energy Efficiency Features of the Intel Alder Lake Architecture. In *Proceedings of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3629526.3645040

1 INTRODUCTION

The continuous evolution of processors requires vendors to translate ever-growing transistor budgets into performance improvements, e.g., by including more functional units, memory controllers, input/output (I/O) interfaces, graphics processing units (GPUs), and caches. This trend also increases complexity, which cannot be fully hidden from the operating system (OS) or application domains. Issues like *where to place threads* if cores have different frequency ranges or architectures, or *where to perform a task* that might be hardware-accelerated cannot be decided on a hardware level. Moreover, performance improvements need to be achieved within a limited power envelope with energy efficiency as a first order design goal. Introduced power saving techniques, however, can contradict OS and applications performance assumptions.

Several processor vendors offer heterogeneous processor architectures, such as ARM's big.LITTLE or Apple M1, combining high-performance and power-efficient cores. Intel's first such architecture, Alder Lake, integrates different core architectures and various accelerating components. This work presents an architecture overview of Alder Lake and an in-depth analysis of its power efficiency properties and techniques. For example, this includes frequency scaling of different components, idle states and their latencies, integrated energy measurement capabilities, and recently introduced processor feedback interfaces and OS integration.

2 BACKGROUND AND RELATED WORK ON ENERGY EFFICIENCY MECHANISMS

Dynamic Voltage and Frequency Scaling (DVFS) describes the ability of a processor to change frequencies and voltages at runtime as a trade-off of power and performance. Contemporary processors have multiple frequency and voltage domains, e.g., one for each core and one for uncore components. The decision to change frequencies is influenced by multiple factors: first, the allowed frequency or frequency range set by the operating system, which can be influenced by the user [22, 34]; second, the internal control mechanisms, which uses one of the allowed frequencies [16, Section 14.4]; and third, protection mechanisms to prevent overheating and a high power consumption [16, Section 14.10]. After the decision is made, frequencies are not changed instantaneously, but only with some delays to change voltages as well. While the information about the duration of a frequency change can be communicated by the hardware to the OS in ACPI P-state tables [32, Section 8.4.5], these values are not necessarily correct [21]. Moreover, internal control mechanisms delay the decisions further [6, 28].

DVFS has been used by researchers to optimize the energy efficiency of programs using two different approaches: First, the region-based tuning of frequencies uses the characteristic of a code region to lower the frequency of components when they are not used. One example is the reduction of processor core frequencies when code is memory-bound, as described, for example, by Kumaraswamy et al. [19] and Vysocky et al. [33]. The second approach applies to parallel applications where the non-critical paths can be slowed down to reduce the energy consumption of the cores that execute them. This has been shown for example by Rountree et al. [25]. Characteristics of frequency transitions have been studied by Mazouz et al. [21]. We described additional details in [6, 28, 29].

Idle States are hardware power saving mechanisms that can be used by the OS or the hardware to switch off the clock (*clock gating*) or voltage (*power gating*) of a part of the processor that is not actively used to lower power dissipation. Operating systems typically use instructions like `hlt` or `mwait` to let CPUs idle [23]. Processors can also employ idle states for whole packages, including cores and uncore components [26]. Parts of a processor core can also be disabled when they are not used [28]. However, idle mechanisms introduce latencies when re-enabling the idling components. Characteristics of idle state transitions have been studied by our previous work for various architectures in [8, 26, 28, 29].

Modern high performance processor architectures typically operate under thermal and/or power constraints when fully utilized [7, Sec. 1.5]. Consequently, processors are equipped with mechanisms for **Power Limiting and Thermal Protection** to enforce operation within the given constraints. Intel introduced the Running Average Power Limit (RAPL) [16, Sec. 15.10], which aims to maximize performance while ensuring safe operation. RAPL also provides energy measurement data that can be read from counters for certain power domains. Monitoring power for thermal protection is also available on other platforms such as AMD [4, 29] and IBM [31]. In previous work we detailed accuracies and other properties of processor and platform power measurement interfaces [4, 6, 28, 29, 31]. Processors can also use clock modulation to lower power consumption for thermal protection. Here, parts of a processor are periodically clock gated for a certain timeframe to lower power dissipation [27]. Another approach forces cores into idle states periodically, including all benefits and costs discussed earlier. This concept is called Hardware Duty Cycling (HDC) on Intel processors [16, Section 15.5].

3 ALDER LAKE PROCESSOR ARCHITECTURE AND TEST SYSTEM

The Intel Alder Lake processor family is a heterogeneous architecture that can include the following computing components: Up to eight performance cores (P-cores) using the Golden Cove architecture, up to eight efficiency cores (E-cores, located in two modules with four cores per module) using the Gracemont architecture, an Intel Xe Gen 12.2 GPGPU with up to 96 execution units (EUs) and a Gaussian-Neuronal-Network-Accelerator (GNA Version 3).

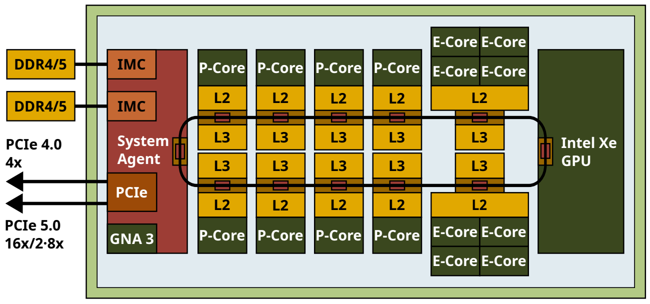


Figure 1: Block diagram of the Intel Core i7-12900K processor

The processor cores are connected via shared caches[1]. To transport data from and to the processor, it supports 16 PCIe 5.0 and 4 PCIe 4.0 lanes, Thunderbolt 4.0, Direct Media Interface (DMI) 4.0 and memory controllers for DDR4 and DDR5. The P-, H-, and U-processor line also include an Image Processing Unit (IPU), which provides support for camera functions (e.g., white balance and color matching). A Volume Management Device [11, Section 2.8] adds hardware RAID support below the OS level. The specification of Alder Lake processors offer a range of configurable parameters for thermal management / power control. Four power limits (PL1 - PL4) define increasing thresholds, starting with PL1 as the average power over long time, up to PL4 as a limit never to be exceeded (see [11, Section 4.1.1]). From these, PL3 and PL4 are disabled by default. When a system (platform) power measurement is available, platform power limits (PsysPL) can further enforce thermal limits beyond the scope of the processor. PL1 and PL2 limitations can also affect main memory accesses, as [13, Section 3.3.13] hints.

Our test system hosts an Intel Core i9-12900K processor with 8+8 processor cores and 32 GPU EUs with a TDP[2] of 125 W. Appendix B lists the full test systems specifications. Figure 1 shows a block diagram of the processor. The Intel powercap kernel module reports 4 kW for the `long_term` and `short_term` RAPL constraints (`power_limit_uw`). According to settings regarding Temperature Targets [13, Section 3.3.28], the processor throttles at 100 °C and the fans are engaged at 80 °C. While the interfaces for Hardware Duty Cycling are listed in [17, Table 2-39], the feature itself is not available on our system according to `cpuid`, and the MSRs are not accessible. We measure power consumption on the AC side using a ZES LMG450 power meter and collect data externally using MetricQ [9] for processing.

4 DYNAMIC VOLTAGE AND FREQUENCY SCALING

Alder Lake processors have clock domains for: cores, the GPU, the memory controller, the system agent, and the uncore including L3-slices and ring. Different interfaces can be used to change core frequencies: The operating system can use model-specific registers (MSR) with the Enhanced Intel SpeedStep Technology [11, Section 2.4.8] [16, Section 14.1]. While these MSRs can be accessed

[1]All cores share the last level cache (LLC / L3), a set of four E-cores shares a mid-level cache (MLC / L2)

[2]The acronym TDP originates from Thermal Design Point [24] or Thermal Design Power but is now also referred to as Processor Base Power [13, Table 1].

per CPU[3], Intel states that *"all active processor IA cores share the same frequency and voltage"* [11, Section 2.4.8]. We validate this in Section 4.1. Alternatively, the processor can control core frequencies transparently using Hardware-Controlled Performance States (HWP) [16, Section 14.4], also known as Intel Speed Shift Technology [11, Section 2.4.10]. This can still be influenced by the OS by regulating the minimal and maximal allowed frequency and the preference for performance or power saving. Since frequency control is implemented with MSRs, a remote core's frequency can only be changed by interrupting that core's work. To cover this issue, the operating system can use a new mechanism Remote Action Request (RAR) [11, Section 2.4.16]. With RAR, a HWP request can be broadcast to all cores of the system [2]. However, on our system, the RAR information register [2, Table 4-1] is not accessible. We describe how long it takes a core to change its frequency in Section 4.2. Frequencies can be increased above the nominal frequency using Turbo mechanisms [11, Sections 2.4.5, 2.4.7, 2.4.10], within the given thermal and power limits. One of the reasons for a high power consumption of processors and a possible reduction of frequencies is the execution of compute-intense instructions. On server processors specific frequency bands are applied when such instructions are used [6, 28]. In Section 4.3, we check whether such mechanisms are also present in the Alder Lake architecture.

The frequency of uncore components is regulated by the processor but can also be influenced by the operating system [17, pp. 2-332f], which exposes this functionality with the *uncore-frequency* driver. Previous work showed that these definitions cannot be considered to be hard limits and that a regulation mechanism will adapt the uncore frequency to workloads on server processors [28]. We cover this for Alder Lake processors in Section 4.4.

The frequency of the integrated GPU can also be changed by the operating system and the hardware using different interfaces [11, Section 3.4.3]. Linux exposes this option to the userspace with the *i915* driver. The datasheet [13, Section 3.3.22] states that compute (slice) components of the GPU have a different frequency than other (unslice) components. We describe the operating system interfaces and the supported frequencies in Section 4.5.

[3]We use *CPU* to refer to a logical OS CPU, which corresponds to a hardware thread.

4.1 Frequency Interdependencies

At first, we check if the core frequencies depend on each other. We run a `while(1);` workload on the tested CPU with a low frequency and set a higher frequency on a different CPU that is either active or idle. An increased frequency of the tested CPU suggests a shared frequency domain. The experiment reveals that all active cores of the processor share one frequency domain running with the highest frequency set for any of these cores. In addition, idling P-core CPUs can increase the frequency of the non-idling CPU of the same core. Likewise, idling E-cores influence other cores in their module.

4.2 Frequency Change Latency

To measure the time until a frequency change is applied, we use the method introduced by Mazouz et al. in [21] and refined in [6, 28]. We pin the frequency of unused CPUs to 800 MHz. The workload starts at the source frequency, waits for a random time between 0 ms and 10 ms, triggers a frequency switch via sysfs, measures the start time and monitors the runtime of a short loop until it fits the performance expected for the target frequency. As soon as that happens, it takes the end time and verifies that the new performance is stable. Unstable outliers are marked as such and filtered from the analysis. Afterwards, it resets the frequency to the initial frequency, verifies it by measuring the short workload and starts over by waiting for a random time to measure the next switch. We take 2×10 000 samples for P-cores and E-cores, respectively.

Figure 2 shows the latencies for P- and E-cores switching from 3.2 GHz to 3.0 GHz and 1.0 GHz to 0.8 GHz depending on the wait time after resetting to the initial frequency. Initially, the transition is fast, e.g., 38.7 µs (P-core, wait time < 2 ms, 3.2⟶3.0 GHz, median). Starting approx. 2.1 ms after the last frequency change, the behavior for P-cores changes and now falls in two categories: For the 3.2⟶3.0 GHz-switch, we see a periodic pattern with a period of ≈200 µs where a new core frequency can be applied at absolute points in time. Such a behavior was reported for server processors in [6, 28, 29]. The latencies for this periodic behavior range from 69.6 µs to 272.2 µs (P-core, wait time > 2.5 ms, 3.2⟶3.0 GHz, 1 % and 99 %-quantiles). In contrast, the 1.0 GHz⟶0.8 GHz transition is now significantly faster without a further time dependent pattern.

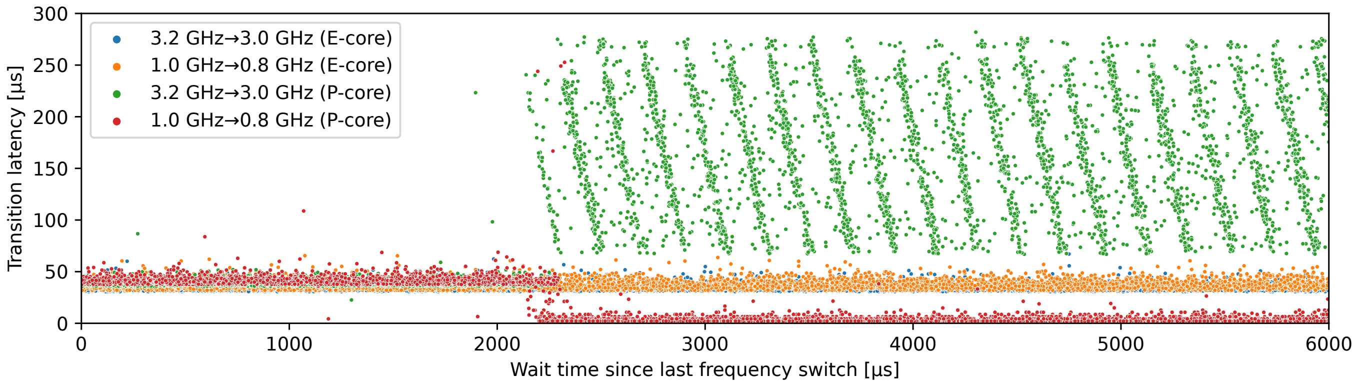
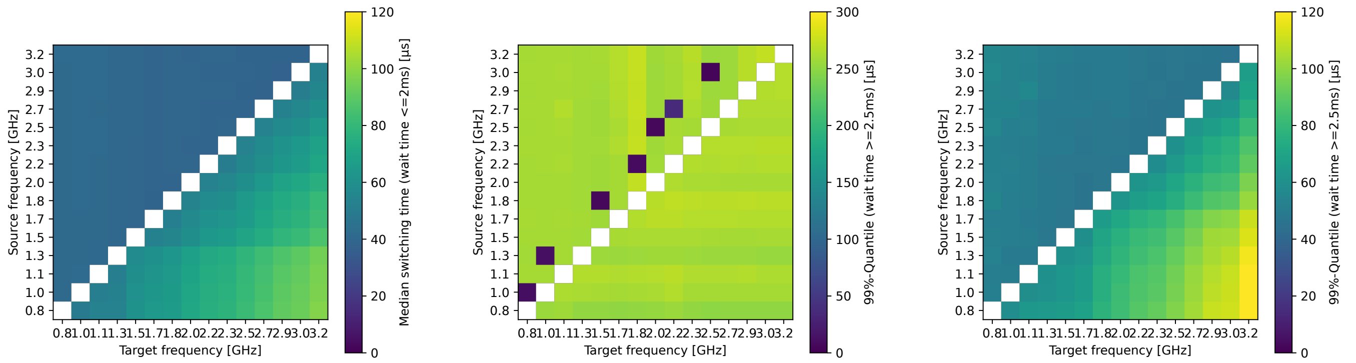


Figure 2: The time until a new frequency is applied depends on the wait time since the last frequency change. The latency for E-cores is independent of that wait time. Latencies on P-cores behave identical to those of E-cores for low wait times. For longer wait times, patterns emerge for P-cores which depend on the start and target frequencies.



(a) P-cores: Median of frequency latencies (<2 ms)

(b) P-cores: 99th percentile of frequency latencies (>2.5 ms)

(c) E-cores: 99th percentile of frequency latencies (>2.5 ms)

Figure 3: Frequency switch latencies, differentiated to short (< 2 ms) and long time (> 2.5 ms) after last switch.

Overall, the transition latencies within the first 2 ms after a previous switch depend only on source and target latencies (see Figure 3a). While lowering frequencies can be done faster without waiting for the voltage to change, increasing frequencies first needs to increase voltages. Figure 3b shows that, for P-cores after 2 ms, most combinations follow the pattern of the 3.2 GHz⟶3.0 GHz transition. However, there are some combinations with a low and constant latency similar to the 1.0 GHz⟶0.8 GHz transition. The latter have some things in common: first, it is always a reduction of frequency and second, the source and target frequencies are relatively close. This could be due to voltages not being changed during the last frequency reset. In addition to that, some target frequencies also show a pattern where a significant number of samples were not valid. We provide more data with the reproducibility package.

These complex effects do not occur on E-cores, as shown in Figure 2 and Figure 3c. Here, the initial pattern of a stable minimal latency and a range of values above the minimum is visible even for higher wait times since the previous frequency switch.

4.3 AVX Frequencies

The perfmon events website lists the event *CORE_POWER* supported by P-cores [10, Event CORE_POWER]. The same event name is used on Intel Skylake processors to monitor processor cycles spent in different frequency bands (standard, AVX, AVX-512 frequencies). On AlderLake P-cores the event lists three different *power licenses*: 1-3, which can be selected using the umasks 0x02, 0x04, and 0x08 respectively. We validate these events by running FIRESTARTER [5] and a `while(1);`-loop while sampling the performance monitoring counter (PMC) every 1 s using `perf stat`. During the `while(1);` workload, the PMC counts cycles if the umask is 0x1. We therefore argue that the umask 0x1 refers to license 0. Running FIRESTARTER workloads with SSE and AVX triggers the PMC to count licenses 1 and 3, respectively.

License 2 is used whenever switching from license 3 to license 0, as determined with `perf record`. Based on the number of cycles spent in license 2 and the duration of a workload, cores presumably spend about 640 µs in license 2 before switching back to license 0. This time correlates with the time to switch back from the AVX-512 to the standard frequency band on Skylake server processors [28, Section VII]. From this, we conclude that it is likely that a mechanism for applying AVX frequency-ranges is implemented in Golden Cove cores. However, we could not see any impact of this mechanism on applied frequencies.

4.4 Uncore Frequency

As in previous products, the uncore frequency is usually regulated within a pre-defined frequency range by an internal control loop [6, 28]. For our processor, the default range spans from 800 MHz to 4700 MHz according to the `UNCORE_RATIO_LIMIT` MSR. By manipulating this register we can reduce the lowest frequency to 400 MHz, but we cannot increase the maximal frequency. In all cases, the uncore frequency is set 200 MHz below the core frequency[4]. The exceptions are the following:

(1) The given bounds are not exceeded in the default case – i.e., at a core frequency of 800 MHz, the uncore still uses a frequency of 800 MHz
(2) Whenever a workload runs on P-cores and E-cores with enabled turbo frequencies, the uncore frequency is reduced to 3.6 GHz–3.7 GHz (200 MHz below E-core turbo). This also overrides the previous exception. Even if the minimum is set to > 3.7 GHz, the uncore frequency stays at the same level.

In some scenarios, users might want to manipulate the uncore frequency manually for energy efficiency reasons, e.g., lowering it during code sections with no offcore accesses. We evaluate the occurring latencies with the methodology introduced in [28]. We measure 1000 transitions for each pair of source and target frequencies and evaluate the median, as depicted in Figure 4. We found the transition latency to depend on the source and the target frequency. While the qualitative pattern is the same as for frequency switches on cores (see Figure 3a and Figure 3c) the quantitative values are different. Lowering the frequency is much faster at about 26.5 µs. Increasing uncore frequencies can take longer compared to core frequency changes.

[4]When disabling the BIOS setting *Ring to Core offset (Down Bin)* the uncore frequency will equal the core frequency. However, by default it is enabled.

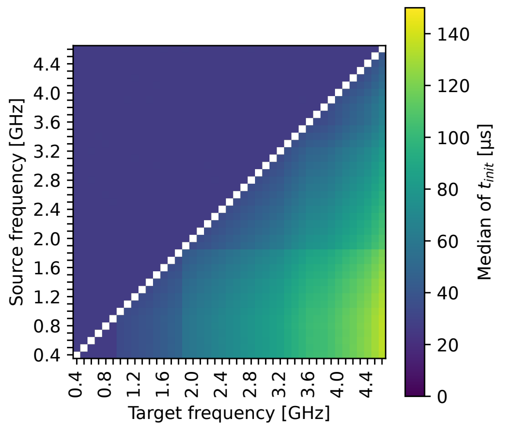


Figure 4: At the lowest core frequency (0.8 GHz), changing the uncore frequency takes between 26.4 µs (clocking down) and 135.0 µs, depending on source and target frequency.

4.5 GPU Frequencies

We use the sysfs interface and `perf stat` events for the i915 device to test valid frequencies on the GPU. The sysfs interface provides three different files to specify the minimal, maximal, and boost frequency in MHz (`gt_{min|max|boost}_freq_mhz`), from which we set the latter two. `perf` lists multiple events for the i915 device, from which we use *actual-frequency* and *requested-frequency*. Both counters only increase when the GPU is actively used by the system. In a first step, we check the supported frequencies and steps. To do so, we increase the requested frequency in 1 MHz-steps while running a matrix multiplication on the GPU. Results show that the actual frequency is always a multiple of 50 MHz. The requested frequency is a multiple of 16.667 MHz. To fit this scheme to the given interface requested frequencies are rounded as follows: First, it is rounded to the nearest multiple of 16.667 MHz. Then it is rounded up to a multiple of 50 MHz.

5 IDLE STATES

Operating systems use idle states to lower power consumption whenever there is no task scheduled on a CPU. Information from ACPI tables, populated by hardware, are the basis to decide which idle state to use [32]. This includes an estimation of the time to re-enable CPUs. As latencies can be several hundred microseconds [26], deep idle states should be disabled for latency sensitive scenarios [3]. In other cases, a lower average power consumption is preferred. While ACPI tables can hold information on projected power consumption, this information is often not set or invalid for idle states. Latency and power consumption additionally depend, for example, on applied frequencies and the activity of other cores.

Alder Lake cores implement three different core C-states: C0 (active), C1 (clock gating, can be combined with DVFS to C1E), and C6 (power gating). As previous processors, the core can autonomously switch from C6 to C1/C1E, which is called auto-demotion. Additional package C-states are used whenever all cores and the GPU reside in a higher C-state (C8, C10). Package C-states can limit the functionality of PCI links and other busses like USB and xHCI [13, Section 3.7f] or flush the L3 cache. During package C-state transitions or whenever a device is still active, PKG C2 can be used where cores are still in a deep idle state, but uncore components can be active. The datasheet [13, Table 8] lists more details about package idle states. We describe the usage of C-states and the power saving potential of our system in Section 5.1 and the time to return to an active state in Section 5.2.

Alder Lake introduces two new idle states[5] (C0.1 and C0.2) that – unlike other C-states – can be entered from userspace [15, Table 4-21]. The two different idle states vary in their wakeup time, power savings, and effects with Simultaneous Multi-Threading (SMT): In C0.2, the wakeup time and power savings are higher and the performance of the second CPU on a core improves. We measure power consumption of C0.1 and C0.2 states in Section 5.1 and analyze the latency to return from these idle states in Section 5.3.

5.1 C-States and Power Consumption

We use different idle states on all cores of the system to determine their power consumption. Our experiments cover the five C-states available through the OS: C0 (POLL), C1E (hlt), C6, C8, and C10, as well as active idling workloads (widely unrolled `NOP`, `PAUSE` for C0, and `TPAUSE` for user idle states C0.1 and C0.2). We use the register `MSR_IA32_POWER_CTL` to disable the transition to C1E. We further test at different core frequencies. To measure the usage of hardware C-States we monitor the idling periods with `perf stat`'s event groups `cstate_core/` and `cstate_pkg/`. Concurrently, we monitor system power consumption using the out-of-band MetricQ framework [9] . We use the mean power measured over an interval of 10 s (11 seconds measured, data from the first second is omitted) with a sampling rate of 20 Sa/s.

According to our measurements, P-cores use the C6 state if requested by the OS. The event counter `cstate_core/c7-residency/` increases if C8 or C10 are requested. E-cores on the other hand use the C6 state, even if higher states are requested. This is surprising, since the OS requests C10 to a high extent according to the usage stats provided by the idle driver. With these core C-states, the highest package C-state that can be used is package C6. Nevertheless, only the package C-states PC2 and PC3 are used, where attached USB and video devices result in a higher proportion of PC2 usage. We provide data, the analysis script and plot for these numbers in the reproducibility data package.

The power consumption resulting from using different idles states is shown in Figure 5. As expected, power consumption in C1 and C0 (including active idle routines) increases with the applied core frequency since power gating is not used. For other C-states, the power consumption is independent of the applied frequency. The usage of C8 and C10 results in the same power consumption, which is consistent with the previous analysis with `perf`. If C8 and C10 are not allowed, the cores use C6 and power consumption increases from 37 W to 43 W. The plot show a particular anomaly: at low core frequency, the power is slightly higher when using C6 than under C1. However, due to a peculiarity that we describe in Section 7, power consumption in these cases is generally inconsistent.

[5]Originally, they were introduced to Atom line with Tremont processors. Now they are also available for performance cores.

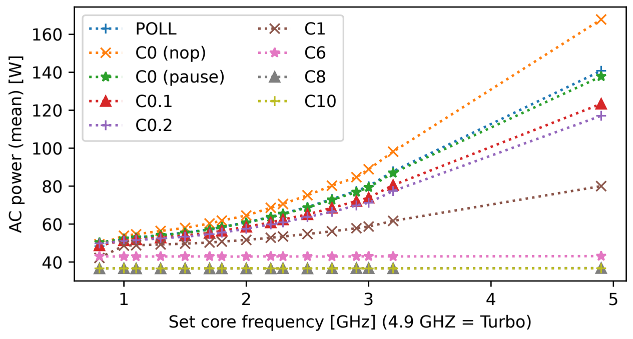


Figure 5: System power consumption depending on used idle mechanism and frequency by cores. 4.9 GHz refers to the usage of Turbo frequencies (measured for P-cores in C0).

5.2 Idle State Latencies

To measure latencies for returning from idle states, we use the methodology from [8]. Here, a thread running on one core (caller) sends a `pthread_cond_signal` to another core (callee), which waits using `pthread_cond_wait`. We measure times as a difference between the Linux kernel events *sched:sched_waking* from the caller and a *power:cpu_idle* to an active state at the callee.

The core C-state C10 is not used (see Section 5.1). Surprisingly, the latencies for C6 and C7 (as initiated by allowing C8 and C10 states) are similar (not depicted). As shown in Figure 6, latencies decrease with an increasing core frequency up to about 2 GHz. Afterwards, there is no clear pattern. The measured times are in the same order of magnitude as Skylake server processors [28].

5.3 User Space Idle State Latencies

In addition to idle states that can only be called from an operating system, Intel implements new instructions that can trigger idle behavior from user space. These include a timed pause instruction and a user-space implementation of `monitor`/`mwait`. With these, threads can indicate to the processor that it should stop fetching and executing new instructions while still being scheduled on the CPU. To limit the duration of these idle periods, operating systems can configure a maximal sleep time in the `UMWAIT_CONTROL` MSR (25 000 cycles on our system).

We check the latency of waking cores from user-space idle states by accessing `umonitor`'ed data. To do so, we setup `umwait` to use a specific state and schedule two threads on two CPUs: a caller and a callee. The callee waits for a volatile date to reach a certain value, then it monitors this value using `umonitor` and waits for it to change using `umwait`. The caller waits for a specific time to take a timestamp using `rdtsc` and change the monitored value. The change causes the callee to wake up and also take a time stamp. We store the difference of both timestamps for the analysis. In Figure 7, we show the distributions for waking up different core types from CPU 0. Here, we see that E-cores (CPU 20) have a lower latency than P-cores (CPU 1). In C0.1 we see an anomaly, where about 8 % of the samples have a higher latency. It seems that the E-cores do not support the C0.2 state: The latencies including the C0.1 anomaly are the same, regardless of the requested state. Also, the spatial distance of cores has an influence on these latencies, but the analysis is out of scope. Data can be found in the reproducibility package.

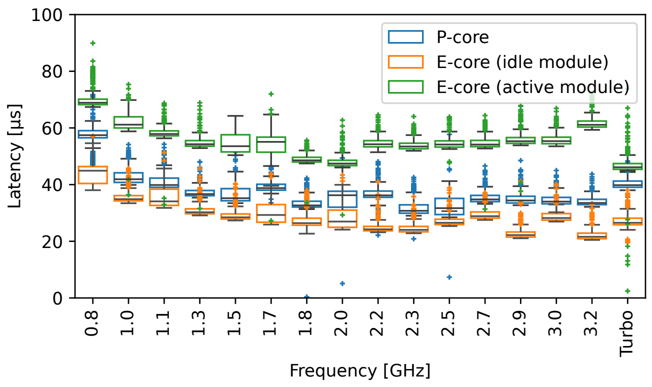


Figure 6: C-state wakeup latencies for C6. For P-cores and E-cores depending if all cores in it's module are idle

6 EFFICIENCY OF IMPLEMENTED COMPUTE-ARCHITECTURES

The two different core architectures and multiple accelerator architectures of Alder Lake processors each implement their own performance and power profile. According to [18, Section 2.2.1] *P-cores provide single or limited thread performance, while E-cores help provide improved scaling and multithreaded efficiency* (see also [18, Section 2.3, Section 4.1]). To support an efficient usage of this heterogeneity, Alder Lake provides a feedback interface for the operating system that gives information about the recently executed workload, distinguishing four classes: *Non-vectorized integer or floating-point code, [...] vectorized code [...], Intel [Deep Learning] Boost code, Pause [...] dominated code* [18, Section 2.2.2.1]. We evaluate this interface in Section 6.1. Intel also implements a Hardware Feedback Interface (HFI), which describes the performance and energy efficiency of each available CPU [16, Section 15.6]. This interface can, e.g., be used by the OS to select specific cores for different workloads. The entries *"may change at runtime as a result of changes in the operating conditions of the system or the action of external factors"* [1], which then can be used for scheduling decisions.

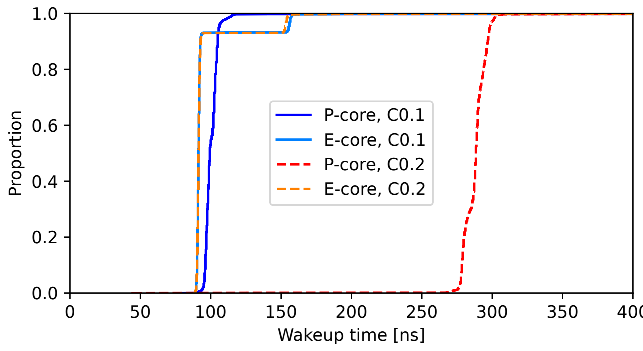


Figure 7: User space idle state wakeup latencies (ECDF)

On Alder Lake processors, Intel implements an Intel Xe Gen 12.2 GPU [13, Section 9.1.1], which can also be used to accelerate computing. Section 6.2 compares performance and energy efficiency of BLAS routines on processor cores and the GPU. Shared memory resources like main memory can have a strong influence on the energy efficiency of workloads [6, 28, 29]. Unfortunately, an analysis of this topic is out of scope for this paper.

6.1 Intel Thread Director

The OS can use the Thread Director interface to observe and schedule threads to cores that are deemed efficient for their workload. Intel lists typical workloads that represent the four classes in [18, Section 2.2.2.1]. We use these workloads in a loop running on all CPUs of the processor. Then, we enable the mechanism via the MSRs `HW_FEEDBACK_CONFIG` and `HW_FEEDBACK_THREAD_CONFIG` and poll the `THREAD_FEEDBACK_CHAR` MSR while executing the workloads. Finally, we compare the provided mapping to the executed workload. E-cores never provide valid samples (bit 63 is set to 0). P-cores can correctly identify classes 0, 2, and 3, but map the class 1 workload to class 0. As the information is missing on E-cores, we doubt that the OS can effectively use this interface to manage scheduling decisions, e.g., migrating unsuitable threads await from E-cores.

6.2 Comparison of Performance and Efficiency

To compare the efficiency of P-cores, E-cores, and GPU, we use three different BLAS functions: dot, sgemv, and sgemm. We vary the frequency of the respective computational unit(s) across the set of specifically selectable frequencies, but not the full turbo range. Thus, the processor runs significantly under its power budget. For GPU workloads, core frequencies are set to 3.2 GHz. The problem sizes are set such that data does not fit in caches. We use the Intel Math Kernel Library (MKL) implementations of sdot, sgemv, sgemm with $N = 1G, 40k, 10k$ respectively. Specific core types are selected with `taskset` for CPU and OpenMP target directives for GPU whereas MKL controls the final number and distribution of threads. We use 100 repetitions for sdot/sgemv and 10 repetitions for sgemm to achieve stable power consumption, but use the median of power samples to avoid impact from initialization and measure the overall execution time to compute the floating-point performance.

Figure 8 shows the resulting performance depending on processing and uncore frequencies. First, we focus on P-core performance at a variable uncore frequency (default). Due to the low arithmetic intensity, the performance of sdot and sgemv is dominated by memory accesses with diminishing benefits from high core frequencies. At default uncore settings, the core frequency only indirectly affects performance via the variable uncore frequency. E.g., P-core frequencies of 800 MHz and 1000 MHz show the same performance since both imply an uncore frequency of 800 MHz as described in Section 4.4. We validate this assumption with the results from a fixed uncore frequency of 3.2 GHz. On the E-cores and the GPU, the uncore bottleneck does not apply and the performance does not appear to saturate in a memory-bound configuration. The sgemm-kernel is purely compute bound for all architectures and frequencies with a performance growing linearly with the core frequency. While we executed the GPU kernels from different host core types and at different uncore frequencies, both performance and power consumption only depends on the GPU frequency.

P-core configurations exhibit the highest overall power consumption, except at the lowest core frequencies. Due to their higher turbo frequencies, P-core power can be even higher, which is not covered by this benchmark. However, due to their superior performance, they are still the most energy-efficient choice for sdot and sgemv with optimal core frequencies around 1800 MHz at default uncore frequency. For sgemm, the GPU at its highest frequency is the most efficient. This efficiency considers the total system energy for the given hardware allocation when exclusively using one architecture. The results are therefore influenced by the power consumption of the common resources, e.g., memory and fans and would differ with a different hardware allocation.

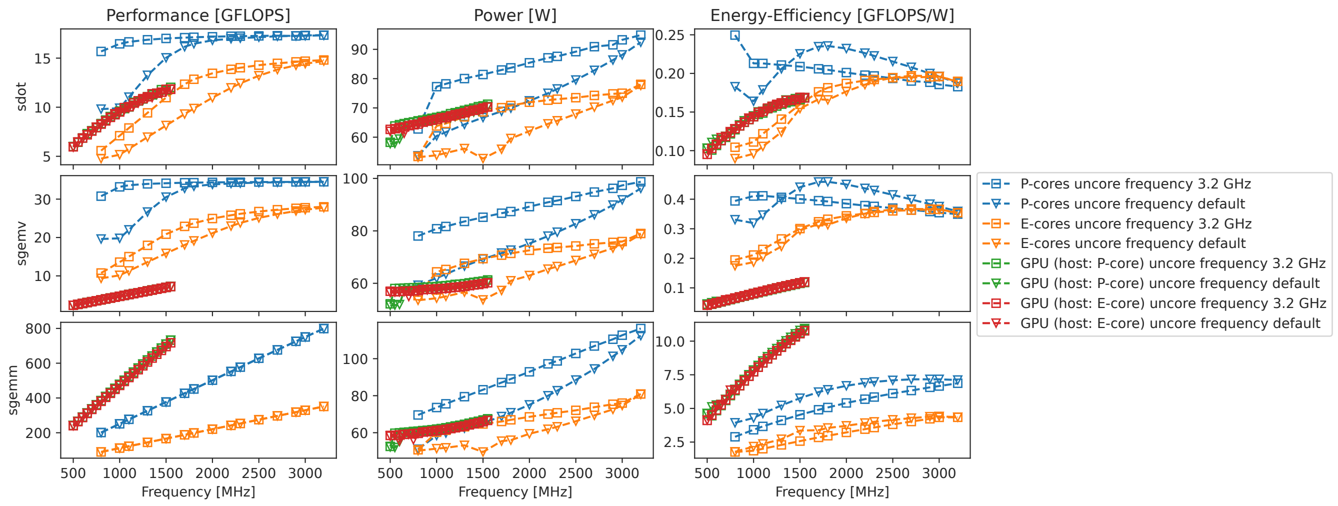


Figure 8: Performance, power and efficiency for different compute kernels on different compute architectures.

7 MONITORING ENERGY-EFFICIENCY-RELATED INFORMATION

The Running Average Power Limit (RAPL) [16, Section 15.10] provides enerhy counters since the Intel Sandy Bridge architecture. While early implementations used models, which were not necessarily accurate [4], newer server processors use physical measurements [6, 28]. The usual update rate of RAPL is 1 ms but can get as low as 50 µs for the PP0 domain (processor cores) on desktop processors. This can be used to retrieve processed data in a side-channel attack [20]. While this can be fixed by limiting access to RAPL counters from the OS side, Intel also implements a filtering technique via the `ENERGY_FILTERING_ENABLE` [17, Table 2-2] entry in an MSR. This filtering adds random noise to the reported values and can only be disabled with a system reset [14]. We analyze the temporal granularity and the filtering feature in Section 7.1

Power measurements for Alder Lake Processors can benefit from the VCCIN AUX IMON Feature to achieve *"more accurate package power reporting and better accuracy"* [11, Section 3.3]. This affects the enforcement of package power limits, but most likely also the accuracy of measuring package power consumption via RAPL. We analyze the accuracy of RAPL measurements in Section 7.2.

Another monitoring infrastructure is the Platform Monitoring Technology (PMT) or Telemetry Aggregator [11, Section 2.6.3], which records metrics of the processor out-of-band and was introduced with Tiger Lake. The data is made available to user space with the *intel_pmt_telemetry* driver and can include information about energy-related features. The definition of the encoded data is not documented. However, an Intel code repository mentions that the interface passes thermal, voltage and frequency information for Sapphire Rapids[6]. The description also notes that some information might be only available under a non-disclosure agreement. In Section 7.3, we analyze the information available on our system.

This study focuses on package (PCKG) and core (PP0) counters, since the DRAM domain reports 0 and the uncore component domain (PP1) reports 0 if the GPU [16, Section 15.10.2] is not used.

[6]https://raw.githubusercontent.com/intel/Intel-PMT/73cfa682/xml/SPR/OOBMSM/CORE/spr_aggregator.xml

7.1 Filter and RAPL Granularity

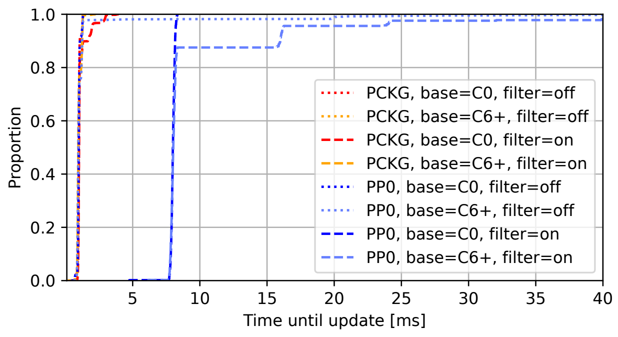
To measure the update rate of RAPL counters, we continuously poll the MSR from CPU 0 for 5 s. We take timestamps before reading the MSR and — if the measured energy changed — we store timestamp, read data, and the number of reading attempts since the last change. As Figure 9a shows, the temporal granularity depends on the monitored domain and the usage of the filter. For the PP0 domain, the temporal granularity is 8 ms if the filter is enabled. For all other cases, the granularity is about 1 ms. If the power consumption is too low for the energy in the MSR to increase, it will be reported in the next available moment (i.e. 1 or 8 ms later). Figure 9b shows effects of the filter on the reported energy. Package power is mostly the same as core power with an offset of about 1.4 mJ, i.e. 1.4 W over 1 ms when idling cores execute the C0 polling routine. If C6 is used, for more than 60 % of the samples, this difference is also visible. However for some samples, the reported package power is the same as the core power. This could hint at a mechanism, which takes energy of uncore components into account only if it is above a certain threshold.

Enabling the filter leads to significant changes: For the package power consumption, a lower energy is reported often. The values are mostly on par with PP0 without a filter for any C-state condition used. For core power consumption, the reported energy increases, as the time frame increases as well. However, the average power consumption over time does not change significantly.

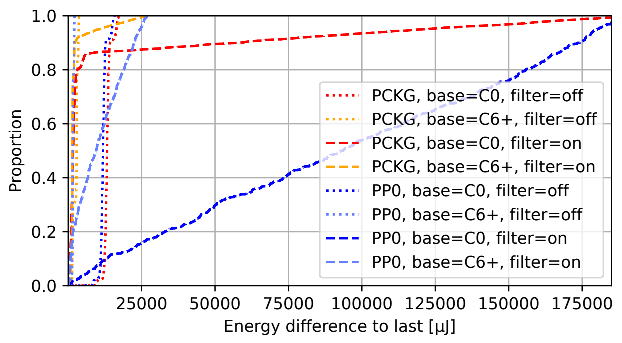
7.2 Accuracy of RAPL Measurements

To further understand the impact of the filter, we measure data-dependent power consumption with and without the filter enabled. To that end, we use run the `vxorps`[7] instruction with different number of set bits on all P-cores with nominal frequency and measure the energy with RAPL. Details of the method are explained in [29]. The experiment measures the `vxorps`-loop 1000 times with 0/50/100% of the bits being set (the operand weight). The result is then split according to the operand weight (defining color) and

[7]We chose vxorps for two reasons: it can be used in encryption algorithms (XOR cipher) and to be comparable to other architectures [29].



(a) Distribution of time between updates: With an enabled filter, the PP0 domain only provides updates every 8 ms, otherwise RAPL values are updated every 1 ms. If the load is too low, some updates might be skipped, e.g., the next update for PP0 and an enabled filter is at 16 ms.



(b) The minimal increase of a measurement is 1 energy unit of 61.035 µJ. Enabling the filter leads to a significant influence on measurements for the PP0 domain and a measureable influence on PCKG measurements.

Figure 9: RAPL information at 2.3 GHz core frequency, which is high enough to enable energy counting on the PP0 domain.

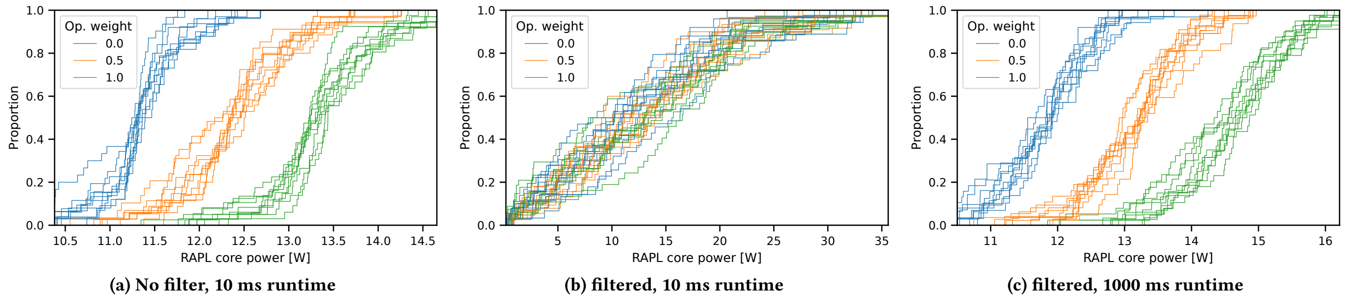


Figure 10: RAPL power samples for measuring a `vxorps`-loop that runs for a given time with a given proportion of set operand bits (operand weight). Each line represents an empirical density function of 100 samples.

split each of these sets into 10 different subsets (to understand repeatability of the effect) that we plot in Figure 10. While the influence of data weight on power consumption is clearly exposed with 100 10 ms-samples without a filter, it cannot be reconstructed with enabled filtering. However, a 100-fold increase of the monitoring time still shows the data-dependent aspect of computations with a similar clarity. This shows that, given enough time, the filter does not prevent side-channel attacks via RAPL monitoring.

On a broader scale, we evaluate the RAPL implementation on our test system using a synthetic workload generator comparing RAPL values (average power derived from energy and time) with the external measurement of average AC power. The reference measurement covers a different domain, including power supply unit (PSU) losses, memory, and other off-chip components of the system. While a direct RAPL measurement error cannot be determined, this approach can be used to expose inconsistencies and systematic errors. The workload generator uses multiple microkernels that stress different components, including a focus on computation and memory. We run each kernel in a large number of configurations, varying core frequency, number of threads, and thread distribution across P-cores, their SMT-threads and E-cores. Each configuration runs for 30 s to prevent timing effects and focus on energy/average power (cf. [4]).

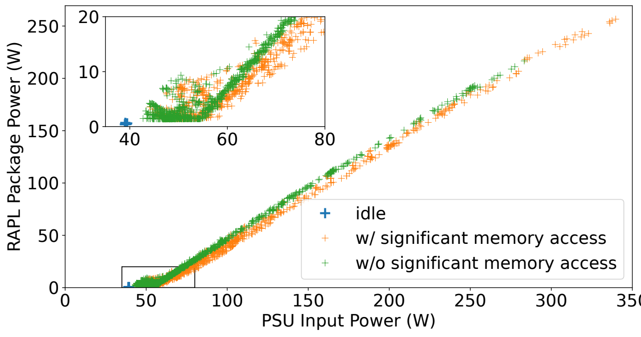


Figure 11: Comparison of power consumption measured by RAPL for the package and a full-system reference measurement. Each point represents the average power within a 30 s interval of the same workload configuration.

Figure 11 plots average AC power against RAPL package power, where each point represents one configuration of kernel, core frequency, and thread configuration. There is a consistent correlation between the two, with two exceptions. First, kernels that utilize the memory subsystem significantly, appear more noisy and report a lower RAPL package power for similar PSU input powers. This can be explained by RAPL not supporting the DRAM domain on this system. While this does not necessarily indicate that RAPL values are wrong, it confirms that RAPL package power alone cannot be used to accurately model total system power consumption, e.g., for energy optimization. The second abnormality occurs at low power consumption, where RAPL and the reference measurement diverge strongly. All kinds of workload kernels and thread configurations, including configurations of P-core and E-core usage exhibit this weak correlation. However, this only affects low frequency configurations: At nominal frequencies, a single thread executing any kernel uses more power than the abnormal cluster.

Figure 12 shows power measurements of different low-power computations with gradually increasing number of threads. While PSU power generally increases with the number of threads, it becomes noisy at 11 active threads with a significantly reduced average. Contrary, the reported RAPL power remains relatively constant at approx. 1.5 W for up to 10 active threads. Later, it increases with substantial noise for configurations where AC power is reduced and noisy Finally, it follows a more consistent pattern at higher power consumption configurations. This behavior is reproducible and we observed similar patterns for other workloads and low core frequencies at different thread count thresholds. We measured several hardware counters (core and uncore frequencies, instructions, power licenses), core temperatures, and workload utility. None of these correlated with the anomalies. We were not able to model or provide an explanation why the actual system power consumption and the RAPL measurements behave in such a way at low power.

We did not include GPU workloads in this evaluation, hence the GPU domain reports 0 W. The PP0 domain follows the package domain with a difference of 1.41 W for workloads w/o substantial memory accesses, up to 2.65 W for workloads w/ memory access, and 0.35 W in idle. In summary, RAPL offers plausible energy measurements with the exception of particularly low power configurations and the limitations to the processor itself.

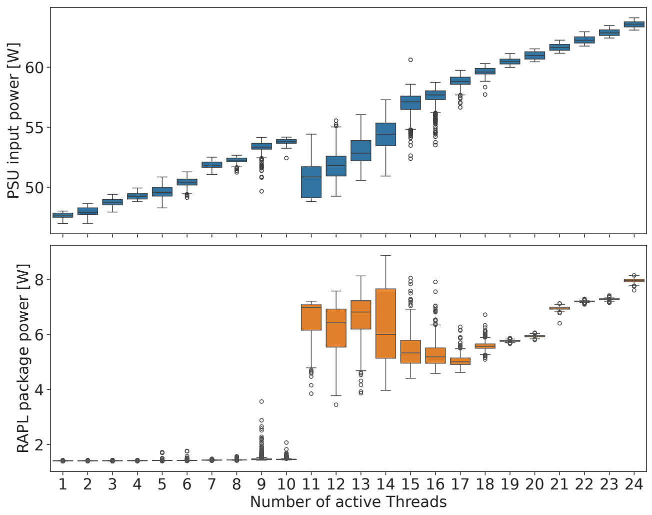


Figure 12: RAPL and reference power consumption sampled at 100 ms / 50 ms intervals respectively. Double precision matrix multiplication kernel at 0.8 GHz running for 60 s each at increasing number of active threads.

7.3 Platform Monitoring Technology

Our test system hosts two Intel PMT telemetry sources with GUIDs 0x85b7c2a0 and 0x409072a0 according to sysfs information. These provide 2752 Byte and 320 Byte of data, respectively. The latter always reads as 0. We run various workloads on all cores or a subset of cores and change hardware properties like core frequencies. We concurrently monitor the data from the main telemetry source by dumping the content of `/sys/class/intel_pmt/telem0/telem` in regular intervals as 4-Byte integers for the follow-up analysis.

In a first analysis step, we check whether the initial guess of data sizes (4 Byte) suits the measured data and distinguish between increasing counters and data that describes a current status. While the former increase over time until they overflow, the latter can change their values more freely. Some rules for the conversion of data types are: A 4-B-date a_i that is increasing towards the maximal integer and increases a_{i+1} on an overflow can be considered an 8-B-date. Any entry representing a current status where the upper 2 Byte are similar to the lower 2 byte represent two 2-B-dates. The same goes for four 1-B-dates. Any entry where a set of 8, 16, or 24 lower bits are not flipped can be shifted by that number of bits.

Some of the most interesting things that we reconstructed are the current frequency of P-cores and E-core-modules (beginning at offset 20), an activity bit-mask for them at offset 76, and their temperature at offset 144. Moreover, the 38.4 MHz signal, which might be related to a clock source of the chipset [12, Section 21], can be read at offset 280, the number of 38.4 MHz cycles a P- or E-core is active are stored at offsets 1840ff, and the uncore clock counts with 200 MHz below the highest frequency at offset 1368. However, some readings of the uncore clock were incorrect (decreasing) during package idle phases. Here it seems that the lower 4 B of the 8 B counter are reset during package idle, while the upper 4 B keep their content. This possible bug also affects other counters, e.g., per core activity related counters starting at offset 1192.

Table 1: Intel PMT data in 85b7c2a0

Starting offset [B]	Assumed content	Size [B]	Data type
20, 24, ...	Frequency of P-cores in 100 MHz	1	Instant.
52, 56	Frequency of E-core-modules in 100 MHz	1	Instant.
76	Bit mask: active P-cores and E-core modules	1.5	Instant.
144, 148, ...	Temperature of P-cores & E-core modules [°C]	1	Instant.
184-212	Core-related, peaks before throttling	2	Instant.
280	Increases with with 38.4 MHz	8	Increasing
1368	Uncore Clock Counter (200 MHz under core frequency) in cycles	8	Increasing
1840-1960	Increases with 38.4 MHz when core is not idling (first p-cores, then e-cores)	8	Increasing

8 SUMMARY AND OUTLOOK

This paper provides a multitude of analyses of power management and energy efficiency features of the first Intel processor generation with heterogeneous core architectures. We found that frequency switch timings differ for P- and E-cores. Only the former show a pattern known from server processors. Even though the architecture does not seem to use AVX frequency ranges, P-cores still support accounting for those. The uncore frequency depends directly on the core frequency and can fall below the set range when cores use Turbo frequencies. E-cores seem not to support some of the idle states of P-cores, namely C0.2 and idle states above C6. Waking cores from deep idle states takes as long as on server processors. User space idle states save power and have a wakeup time in the order of hundreds of nano seconds. The Intel Thread Director fails to identify class 1 workloads and is not present on E-cores. While the integrated GPU can be used to run floating point intense code efficiently, accessing memory seems to pose a bottleneck. When running memory-bound codes, P-cores are most efficient in terms of performance and energy efficiency. Increasing the uncore frequency can increase performance for memory-bound codes at low core-frequencies. The update rates for RAPL are at 1 ms. The filtering functionality can add noise to counter side-channel attacks. Enabling this filter leads to a lower update rate for the core domain. Still, data can be inferred from RAPL readings using longer observation. While RAPL is generally consistent with external measurements, it does not include DRAM and is inaccurate in low power scenarios. The platform monitoring technology provides a sideband measurement of power-related information. We unveil encoded information and describe a possible bug in 8 B-counters.

While not all of these various findings can be generalized beyond our specific system, they serve as a guideline of relevant effects. Moreover, we provide a detailed methodology as well as a reproducibility package [30] to facilitate translating the results to other systems. Some energy aspects are not covered in this paper and remain future work, including the following: In addition to DVFS, the integrated GPU supports various power saving mechanisms [11, Section 3.4] that could be investigated. We also did not analyze the Power Management Integrated Circuits mentioned in [11, Section 3.4] due to a lack of information. Other idle states like S-states or G-states could be investigated further. However, information on their effects are described in [11, 12]. It will also be interesting to look at Sapphire Rapids, which also implements Golden Cove cores, including an analysis of user space idle states, and AVX frequencies.

REFERENCES

[1] Linux Kernel documentation, x86-specific Documentation, Chapter 14: Hardware-Feedback Interface for scheduling on Intel Hardware. URL https://docs.kernel.org/arch/x86/intel-hfi.html. (accessed 2024-02-14).

[2] Intel Corporation. Remote Action Request White Paper, July 2021. URL https://www.intel.com/content/dam/develop/external/us/en/documents/341431-remote-action-request-white-paper.pdf. (accessed 2023-11-17).

[3] Corey Gough, Ian Steiner, and Winston A. Saunders. *Energy Efficient Servers: Blueprints for Data Center Optimization.* Apress, USA, 1st edition, 2015. ISBN 1430266376.

[4] Daniel Hackenberg, Thomas Ilsche, Robert Schöne, Daniel Molka, Maik Schmidt, and Wolfgang E. Nagel. Power Measurement Techniques on Standard Compute Nodes: A Quantitative Comparison. In *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS).* IEEE, April 2013. doi: 10.1109/ispass.2013.6557170.

[5] Daniel Hackenberg, Roland Oldenburg, Daniel Molka, and Robert Schöne. Introducing FIRESTARTER: A processor stress test utility. In *2013 International Green Computing Conference Proceedings*, 2013. doi: 10.1109/IGCC.2013.6604507.

[6] Daniel Hackenberg, Robert Schöne, Thomas Ilsche, Daniel Molka, Joseph Schuchart, and Robin Geyer. An Energy Efficiency Feature Survey of the Intel Haswell Processor. In *IEEE International Parallel and Distributed Processing Symposium Workshop (IPDPSW).* IEEE, 2015. doi: 10.1109/IPDPSW.2015.70.

[7] John L. Hennessy and David A. Patterson. *Computer Architecture : A Quantitative Approach.* Morgan Kaufmann, 6th edition, 2019. ISBN 9780128119051.

[8] Thomas Ilsche, Robert Schöne, Philipp Joram, Mario Bielert, and Andreas Gocht. System Monitoring with lo2s: Power and Runtime Impact of C-State Transitions. In *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, May 2018. doi: 10.1109/IPDPSW.2018.00114.

[9] Thomas Ilsche, Daniel Hackenberg, Robert Schöne, Mario Bielert, Franz Höpfner, and Wolfgang E. Nagel. MetricQ: A Scalable Infrastructure for Processing High-Resolution Time Series Data. In *2019 IEEE/ACM Industry/University Joint International Workshop on Data-center Automation, Analytics, and Control (DAAC)*, pages 7–12, 2019. doi: 10.1109/DAAC49578.2019.00007.

[10] Intel Corporation. Perfmon events Website - Alder Lake. URL https://perfmon-events.intel.com/ahybrid.html. (accessed 2023-11-17).

[11] Intel Corporation. 12th Generation Intel Core Processors Datasheet, Volume 1 of 2. Technical Report 655258, Rev.: 009, Intel Corporation, 2022. URL https://cdrdv2.intel.com/v1/dl/getcontent/655258. (accessed 2022-09-01).

[12] Intel Corporation. Intel 600 Series Chipset Family Platform Controller Hub Datasheet, Volume 1 of 2. Technical Report 648364, Rev 003, Intel Corporation, May 2022. URL https://cdrdv2.intel.com/v1/dl/getcontent/648364. (accessed 2023-11-17).

[13] Intel Corporation. 12th Generation Intel Core Processors Datasheet, Volume 2 of 2. Technical Report 655259, Rev 003, Intel Corporation, April 2022. URL https://cdrdv2.intel.com/v1/dl/getcontent/655259. (accessed 2023-11-17).

[14] Intel Corporation. Software Security Guidance Running Average Power Limit Energy Reporting / CVE-2020-8694 , CVE-2020-8695 / INTEL-SA-00389. Technical report, Intel Corporation, February 2022. URL https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/advisory-guidance/running-average-power-limit-energy-reporting.html. (accessed 2023-11-17).

[15] Intel Corporation. Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, M-U. Technical Report 253667-081US, Intel Corporation, September 2023. URL https://cdrdv2.intel.com/v1/dl/getContent/671241. (accessed 2023-11-17).

[16] Intel Corporation. Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3B: System Programming Guide, Part 2. Technical Report 253669-081US, Intel Corporation, September 2023. URL https://cdrdv2.intel.com/v1/dl/getContent/671427. (accessed 2023-11-17).

[17] Intel Corporation. Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 4: Model-Specific Registers. Technical Report 335592-081US, Intel Corporation, September 2023. URL https://cdrdv2.intel.com/v1/dl/getContent/671098. (accessed 2023-11-17).

[18] Intel Corporation. Intel® 64 and IA-32 Architectures Optimization Reference Manual Volume 1. Technical Report 248966-048, Intel Corporation, August 2023. URL https://cdrdv2.intel.com/v1/dl/getContent/671488. (accessed 2023-11-17).

[19] Madhura Kumaraswamy, Anamika Chowdhury, Andreas Gocht, Jan Zapletal, Kai Diethelm, Lubomir Riha, Marie-Christine Sawley, Michael Gerndt, Nico Reissmann, Ondrej Vysocky, Othman Bouizi, Per Gunnar Kjeldsberg, Ramon Carreras, Robert Schöne, Umbreen Sabir Mian, Venkatesh Kannan, and Wolfgang E. Nagel. Saving Energy Using the READEX Methodology. In *Tools for High Performance Computing 2018 / 2019*, pages 27–53. Springer International Publishing, 2021. ISBN 978-3-030-66057-4. doi: 10.1007/978-3-030-66057-4_2.

[20] Moritz Lipp, Andreas Kogler, David Oswald, Michael Schwarz, Catherine Easdon, Claudio Canella, and Daniel Gruss. PLATYPUS: Software-based Power Side-Channel Attacks on x86. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 355–371, 2021. doi: 10.1109/SP40001.2021.00063.

[21] Abdelhafid Mazouz, Alexandre Laurent, Benoît Pradelle, and William Jalby. Evaluation of CPU Frequency Transition Latency. *Computer Science - Research and Development*, 2014. doi: 10.1007/s00450-013-0240-x.

[22] Venkatesh Pallipadi and Alexey Starikovskiy. The Ondemand Governor Past, Present, and Future. In *Proceedings of the Ottawa Linux Symposium (OLS)*, 2006. URL https://www.kernel.org/doc/ols/2006/ols2006v2-pages-223-238.pdf. (accessed 2023-11-17).

[23] Venkatesh Pallipadi, Shaohua Li, and Adam Belay. cpuidle: Do nothing, efficiently. In *Proceedings of the Ottawa Linux Symposium (OLS)*, 2007. URL https://www.kernel.org/doc/ols/2007/ols2007v2-pages-119-126.pdf. (accessed 2023-11-17).

[24] Efraim Rotem, Alon Naveh, Avinash Ananthakrishnan, Eliezer Weissmann, and Doron Rajwan. Power-Management Architecture of the Intel Microarchitecture Code-Named Sandy Bridge. *IEEE Micro*, 32(2), 3 2012. ISSN 0272-1732. doi: 10.1109/MM.2012.12.

[25] Barry Rountree, David K. Lownenthal, Bronis R. de Supinski, Martin Schulz, Vincent W. Freeh, and Tyler Bletsch. Adagio: Making DVS Practical for Complex HPC Applications. In *Proceedings of the 23rd International Conference on Supercomputing*, ICS '09, page 460–469, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605584980. doi: 10.1145/1542275.1542340.

[26] Robert Schöne, Daniel Molka, and Michael 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.

[27] Robert Schöne, Thomas Ilsche, Mario Bielert, Daniel Molka, and Daniel Hackenberg. Software Controlled Clock Modulation for Energy Efficiency Optimization on Intel Processors. In *Proceedings of the 4th International Workshop on Energy Efficient Supercomputing*, E2SC '16, pages 69–76. IEEE, 2016. ISBN 978-1-5090-3856-5. doi: 10.1109/e2sc.2016.015.

[28] Robert Schöne, Thomas Ilsche, Mario Bielert, Andreas Gocht, and Daniel 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.

[29] Robert Schöne, Thomas Ilsche, Mario Bielert, Markus Velten, Markus Schmidl, and Daniel Hackenberg. Energy efficiency aspects of the amd zen 2 architecture. In *2021 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 562–571, 2021. doi: 10.1109/Cluster48925.2021.00087.

[30] Robert Schöne, Markus Velten, Thomas Ilsche, Mario Bielert, and Daniel Hackenberg. Reproducibility package for paper "Energy Efficiency Features of the Intel Alder Lake Architecture", as submitted to ICPE 2024, March 2024. URL https://doi.org/10.5281/zenodo.10782722.

[31] Hannes Tröpgen, Mario Bielert, and Thomas Ilsche. Evaluating the Energy Measurements of the IBM POWER9 On-Chip Controller. In *Proceedings of the 2023 ACM/SPEC International Conference on Performance Engineering*, ICPE '23, page 67–76, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400700682. doi: 10.1145/3578244.3583729.

[32] UEFI Forum, Inc. Advanced Configuration and Power Interface (ACPI) specification, revision 6.5, August 2022. URL https://uefi.org/sites/default/files/resources/ACPI_Spec_6_5_Aug29.pdf. online at uefi.org (accessed 2023-11-17).

[33] Ondrej Vysocky, Martin Beseda, Lubomír Říha, Jan Zapletal, Michael Lysaght, and Venkatesh Kannan. MERIC and RADAR Generator: Tools for Energy Evaluation and Runtime Tuning of HPC Applications", booktitle="High Performance Computing in Science and Engineering. pages 144–159, Cham, 2018. Springer International Publishing. ISBN 978-3-319-97136-0. doi: 10.1007/978-3-319-97136-0_11.

[34] Rafael J. Wysocki. `intel_pstate` CPU Performance Scaling Driver. In *The Linux kernel user's and administrator's guide*, 2017. URL https://www.kernel.org/doc/html/v4.12/admin-guide/pm/intel_pstate.html. (accessed 2023-11-17).

A ACKNOWLEDGMENTS AND REPRODUCIBILITY

We would like to thank Mario Bielert, Hannes Tröpgen, and the anonymous reviewers for their valuable input. We provide artifacts for reproducibility at [30]. This work was funded by the Bundesministerium für Bildung und Forschung via the EECliPs project under grant no. 16ME0602. It is also financed on basis of the budget passed by the Saxon State Parliament in Germany.

B TEST SYSTEM SPECIFICATION

The test system we used is specified in Table 2.

Table 2: Test system details

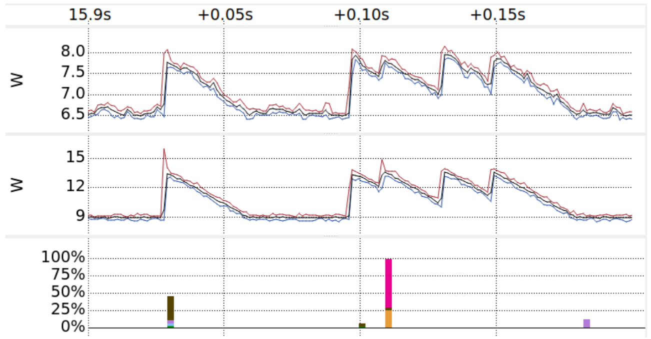
Processor	Intel Core i9-12900K
Performance cores / P-cores / Golden Cove	
Nr. cores / hardware-threads	8 / 16
Frequency range (selectable)	0.8 GHz to 3.2 GHz
Turbo frequency	up to 5.1 GHz*
Efficiency cores / E-cores / Gracemont	
Nr. cores / hardware-threads	8 / 8
Frequency range (selectable)	0.8 GHz to 3.2 GHz
Turbo frequency	up to 3.9 GHz*
Intel® UHD Graphics 770	
Nr. Execution Units (EUs)	32
Frequency range (selectable)	0.3 GHz to 1.5 GHz
Turbo frequency	up to 1.55 GHz
Uncore frequency scaling (UFS)	0.8 GHz to 4.7 GHz
Hardware Performance States (HWP)	disabled
RAPL Power Limit	4095 W
RAM	2 × 16 GiB DDR5-4800
Motherboard	Gigabyte Tech. Co. Z690 UD
Operating system	Ubuntu 22.04
Kernel version	5.19.1 & 6.2.0
Power meter	ZES LMG450
Accuracy	0.07 % + 0.25 W

*or 5.2 GHz on one core or with Turbo Boost Max Technology 3.0

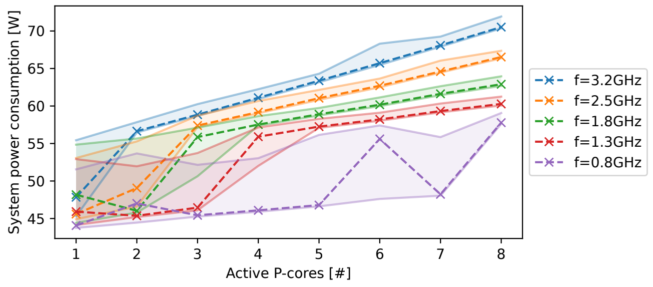
C ADDITIONAL DATA ON POWER MONITORING

On an idling system, additional noise can increase the average power consumption significantly, as we show in Figure 13a. Often, when a CPU on an idling system gets active, the processor power consumption (monitored with an LMG670 and 20 kSa/s) increases from 9 W to 13.3 W (with spikes of more then 15 W). Simultaneously, the power consumption of the 5 V rail increases from 6.5 W to 7.9 W. This increased power consumption only decreases gradually and affects the power consumption of the system even after the CPUs are already back in an idling phase.

We see the multiple power levels mentioned in Section 7.2 for an idling system as well where some cores use the POLLing idle routine instead of the default C10. Figure 13b shows the minimal, maximal, and median power on an idling system with a number of cores POLLing instead of using C10 and an uncore frequency of 3.2 GHz for different core frequencies. The power data is retrieved over a 10 s time period with `metricq-summary`. System power consumption grows linearly with the number of used cores if the system power consumption is above 54 W (with some noise induced outliers). There is a gap if the frequency and number of cores are too low to reach this threshold. There seems to be another linear relation at the minimal values, just on a lower level. Median and average are located randomly between these two lines.



(a) Unused cores can significantly increase power consumption once they become active due to OS noise. Vampir visualization of a lo2s trace of the idling test system. Top: Power consumption (5V rail). Mid: Power consumption (12V processor). Bottom: activity of CPUs. A short usage of CPUs increases the power consumption from 9 to 15 Watt, which is then reduced only gradually. Not only CPU activity can lead to power spikes, e.g., at offset +0.15s there is a spike without any CPU core being active.



(b) System power consumption, depending on the number of cores in OS poll loop and the core frequency. Minimal, median and maximal power values plotted as lines. The uncore frequency is set to 3.2 GHz.

Figure 13: Detailed look at different power measurement aspects.