Scheduled Disclosure: Turning Power Into Timing Without Frequency Scaling

Inwhan Chun, Isabella Siu, Riccardo Paccagnella
Carnegie Mellon University

***Abstract*—Power side-channel attacks are seeing a resurgence of interest in computer security research. An emerging class of these attacks exploits remote methods to monitor power consumption—most notably by observing power-dependent CPU frequency variations. However, existing methods have only been demonstrated on (older) x86 CPU architectures where frequency scaling is the primary—if not only—mechanism utilized to keep the system within safe operating conditions. It remains unclear whether remote power side-channel attacks are still feasible on modern x86 CPU architectures with additional, more sophisticated such mechanisms.**

We demonstrate that not only do remote power-side channel attacks remain feasible on modern x86 CPU architectures, but that they are also *more effective and work even in the absence of frequency side-channel leakage*. Our attacks take advantage of Thread Director, a hardware optimization that provides scheduling "hints" to enhance performance and energy efficiency on modern Intel processors. We demonstrate that these hints depend on the processor's power consumption, leading to power-dependent scheduling behaviors—such as variations in the number of active cores—that can be observed purely from software and even via remote-timing analysis. We show the efficacy of our attacks by leaking keys from constant-time cryptographic code (5× faster than prior attacks on older x86 CPUs) and mounting cross-origin pixel stealing attacks.

1. Introduction

Power side-channel attacks are seeing a resurgence of interest in computer security research. Traditionally, these attacks required access to power measurement equipment (e.g., oscilloscopes) located in physical proximity to the victim device [1], [2], [3], [4], [5]. In the past few years, however, researchers have demonstrated new classes of power side-channel attacks where attackers infer power consumption *remotely*, without any physical access. These remote attacks have been used to leak cryptographic keys [6], [7], [8], [9], [10], [11], revive pixel stealing attacks [11], [12], fingerprint websites [12], [13], [14], break KASLR [9], [14], [15], and mount Meltdown- and MDS-style attacks [16].

To date, two methods have been shown to enable remote power side-channel attacks on x86 CPUs. The first method abuses the Running Average Power Limit (RAPL) interface [7], [8], [13], [15], [16], [17]. This method allows directly measuring the processor's power consumption from software but has been mitigated by restricting and adding noise to RAPL measurements [18], [19]. The second method—dubbed *Hertzbleed*—indirectly measures the processor's power consumption by monitoring power-dependent CPU frequency variations that occur when the processor exceeds power or thermal limits [9], [10], [11], [16].

However, the effectiveness of the Hertzbleed method has only been demonstrated on older x86 CPU architectures, where frequency scaling is the primary—if not only—mechanism utilized to keep the system within safe power and thermal limits. In contrast, many modern x86 processors use multiple, more sophisticated such mechanisms, where, as we show in Section 4.2, frequency scaling is sometimes only used as a last resort. It remains unclear whether remote power side-channel attacks are still feasible on these systems and whether power-management mechanisms beyond frequency scaling introduce new leakage channels.

In this paper, we demonstrate that not only do remote power-side channel attacks remain feasible on modern x86 CPU architectures, but that they are also *more effective and work even in the absence of frequency side-channel leakage*. Our discovery takes advantage of Thread Director, a hardware optimization that helps the operating system (OS) schedule threads to the "most appropriate core" (e.g., P-core or E-core) on recent Intel processors [20], [21]. We demonstrate that, under certain circumstances, Thread Director's scheduling "hints" depend on the processor's power consumption. Moreover, these hints result in power-dependent scheduling behaviors—such as variations in the number of active cores—which can be observed purely from software and even via remote-timing analysis.

We start by reverse engineering, for the first time, the Thread Director optimization on a modern Intel Meteor Lake processor. First, we find that Thread Director assigns one of four *class IDs* to each running thread based on the instruction mix it is running at the granularity of microseconds. Second, we show that when the processor is power constrained and the CPU cores are unable to run above certain frequencies, Thread Director gives *idling hints* to the OS, recommending to use a reduced set of cores rather than further lowering the CPU frequency. These hints depend on the power consumption of the whole processor, including the CPU and the integrated GPU. Finally, we find that Thread Director's hints depend on the CPU and GPU active states.

We then analyze how the Windows scheduler acts on Thread Director's hints. First, we show that Windows gives threads with different class IDs different priorities on the P-cores. Interestingly, this mechanism deviates from Windows' stated goal of treating same-scheduling priority threads as equal [22]. Second, we show that Windows does not sched-

ule threads on cores with idling hints. This behavior results in the average number of active cores varying depending on the processor's power consumption. Notably, this occurs even when the CPU frequency of all active cores is constant and thus independent of power consumption. Moreover, this behavior is observable via remote timing, since differences in core usage lead to execution time differences.

We demonstrate the security implications of our findings by mounting covert and side-channel attacks. Our covert channel attack exploits the observation that a process can "game" the Windows scheduler into making different decisions solely as a function of its class ID. For example, a receiver thread executing scalar instructions may be scheduled on an E-core when a sender process runs vector instructions; however, if the sender runs busy spin loops, the same receiver thread may instead be scheduled on a P-core. We demonstrate that this mechanism can be abused to mount a reliable covert channel with a channel capacity of up to 16.62 bits per second. Despite its low channel capacity, this covert channel does not rely on the sharing (and probing) of any microarchitectural structure nor the use of a timer; the only requirement is a way for the receiver to identify the core type it is running on, which can be done via Windows' `GetCurrentProcessorNumber` API.

Finally, we demonstrate two examples of side-channel attacks that exploit power-dependent scheduling. Our first side-channel attack targets cryptographic code running on the CPU. Specifically, we reproduce the attack demonstrated by Wang et al. [9] on a constant-time implementation of SIKE. The attack leverages the fact that, when provided with specially-crafted inputs, the power consumption of SIKE's decapsulation depends on individual key bits. The original attack inferred key bits by monitoring CPU frequency variations, which in turn produced observable remote timing differences. Our attack infers key bits by monitoring variations in scheduling behavior, which also produce remote timing differences. Under default system settings, our attack recovers the full key via remote timing in 7 hours, which is $5\times$ faster than the time reported by Wang et al. [9] on older x86 CPUs. Moreover, the attack remains effective even when the frequency of all active cores is constant.

Our second side-channel attack targets browser code running on the GPU. Specifically, we demonstrate a cross-origin pixel stealing attack on the latest version of Google Chrome (as of April 2025). Similarly to prior work from Wang et. al [11] and Taneja et al. [12], our attack exploits the pixel color-dependent power consumption of the GPU when applying SVG filters on top of iframes. Unlike prior work, our attack does not rely on measuring differences in the CPU frequency or the GPU rendering time. Instead, it leverages the fact that when the GPU consumes more power, Thread Director gives more aggressive idling hints, causing CPU threads to be scheduled on a limited set of cores earlier; an attacker can observe this behavior indirectly from Javascript, since running the same operations on different sets of cores takes different amounts of time. The attack takes 7.1 s per pixel, which is on par with prior pixel stealing attacks exploiting remote power side channels [11], [12].

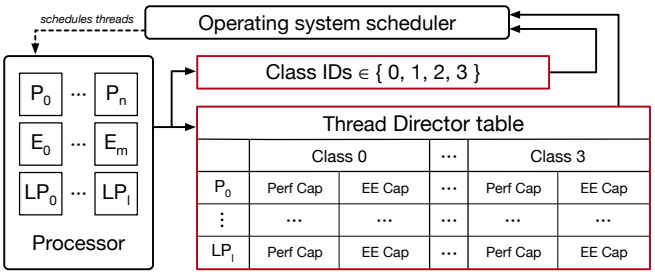


Figure 1. Thread Director assigns a *class ID* to each running thread and provides guidance on the placement of threads to different CPU cores via the *Thread Director table*. On Meteor Lake, this table has one row for each CPU core and two columns indicating the performance capabilities (**Perf Cap**) and energy efficiency capabilities (**EE Cap**) for each class ID. The scheduler can use this information to guide scheduling decisions.

Disclosure. We disclosed our findings to Intel, Microsoft, and Google in November 2024. Intel assessed the vulnerability as a variant of Hertzbleed (aka CVE-2022-24436). Microsoft assessed the vulnerability as "moderate severity," but—as of April 2025—was still deciding whether and how to mitigate. Google (in reference to the pixel stealing attack) stated that they "don't have a comprehensive solution for this bug yet, and won't for a while". They later resolved our and other closely related pixel stealing attacks as "WONTFIX" [23]. None of the vendors requested an embargo.

2. Background

2.1. Heterogeneous CPU architectures

Heterogeneous CPU Architectures, also known as *hybrid architectures*, are architectures featuring multiple types of cores with different microarchitectural characteristics that cause them to run workloads with different performance and energy-efficiency profiles. Despite having been common in mobile systems-on-a-chip for over a decade [24], heterogeneous CPU architectures have only recently been introduced to mainstream PC processors from Apple [25], Intel [26], [27] and AMD [28]. In this work, we study the Intel Meteor Lake heterogeneous CPU architecture [29], which was introduced in late 2023 and uses a combination of performance-optimized cores (P-cores), efficiency-optimized cores (E-cores, organized in clusters of 4), and low-power efficiency-optimized cores (LP E-cores). We use $\texttt{P}_\texttt{n}$ to indicate the n'th P-core and $\texttt{P}_{\texttt{n-m}}$ to indicate P-cores n to m. We use a similar notation to indicate E-cores ($\texttt{E}_\texttt{n}$) and LP E-cores ($\texttt{LP}_\texttt{n}$). When Hyper-threading is enabled, we use $\texttt{P}_{\texttt{i}_{ht0}}$ and $\texttt{P}_{\texttt{i}_{ht1}}$ to indicate the two hyperthreads of a P-core $\texttt{P}_\texttt{i}$. We count cores starting from index $n = 0$.

2.2. Intel Thread Director

Thread Director is an optimization designed to improve task scheduling on Intel's heterogeneous CPU architectures [21]. Specifically, Thread Director gives "hints" to the operating system (OS) to help schedule "the right task to the

right core type" [20]. Thread Director was first introduced in 2021 with the Alder Lake microarchitecture and has been used on most desktop and mobile Intel processors since then [30]. Thread Director consists of two components: (i) a runtime characterization of threads into *classes* and (ii) a hardware feedback interface table (*Thread Director table*).

Workload classification. Thread Director's first component monitors the instruction mix of software threads and assigns a *class ID* to each thread [31]. On Meteor Lake CPUs, this class ID can take values between 0 and 3 and represents the performance difference between running that thread on different core types [30]. Class 0 denotes workloads that perform similarly on P-cores and E-cores (e.g., scalar instructions). Classes 1 and 2 denote workloads that perform better on P-cores than on E-cores (e.g., vector, AI, and accelerated instructions). Class 3 denotes spin loops that do not scale with higher performance. The OS can read the class ID of a running thread from its core's `IA32_THREAD_FEEDBACK_CHAR` model-specific register (MSR) [21].

Thread Director table. Thread Director's second component is the *Thread Director table* (also called the *EHFI table* [21]), which provides guidance to the OS scheduler on the placement of threads among CPU cores. Figure 1 shows what the Thread Director table looks like on a Meteor Lake processor with n P-cores, m E-cores, and l LP E-cores. The table has one row for each CPU core and two columns for each class ID. The two columns contain a *performance capability* and an *energy-efficiency capability* for running a thread with that class ID on each core. These capabilities are numeric values ranging from 0 to 255, where a higher value indicates a higher capability. When it is time for the OS scheduler to schedule a thread, the scheduler can use that thread's class ID and the Thread Director table to dynamically determine which cores are more performant and which cores are more energy efficient for that thread [21]. The Thread Director table is stored in memory, at the physical address indicated in the `IA32_HW_FEEDBACK_PTR` MSR. The hardware can dynamically update the Thread Director table capabilities at runtime and notify the OS via interrupts.

2.3. Processor power management

P-states. Intel CPUs adjust their frequency at the granularity of *P-states*, where each P-state corresponds to a different frequency level in 100 MHz increments [21]. The range of P-states varies depending on the processor model and CPU core type. Each CPU core has a *base frequency* and a *max turbo frequency* [32]. The base frequency indicates the frequency a CPU core should be able to sustain under normal operating conditions without exceeding its Thermal Design Power (TDP). The max turbo frequency indicates the maximum single-core frequency a CPU core can achieve. We use the same P-state naming convention as Linux, where a higher P-state indicates a higher frequency [33]. When Turbo Boost is disabled, a core's highest available P-state corresponds to its base frequency. When Turbo Boost is enabled, a core's highest available P-state corresponds to its max turbo frequency. On Meteor Lake, P-states are managed entirely by the hardware based on workload demands and thermal and power constraints [33], and each core can have its P-state adjusted independently from one another [30].

Throttling. Modern Intel processors use power management algorithms to ensure safe operating conditions after the processor meets certain reactive limits (e.g., thermal or power limits) [10], [34]. In this paper, we define *throttling* as the state a processor enters after reaching these reactive limits. Prior work shows that when the processor throttles (i.e., after it meets reactive limits), the hardware dynamically adjusts the CPU frequency based on the processor's power consumption [9], [10]. The root cause is that Intel processors always try to run at the highest possible CPU frequency given the available power budget, and the available power budget depends on the processor's power consumption.[1]

PP0 and PP1. Modern x86 processors support the Running Average Power Limit (RAPL) interface that can be used to measure an estimate of the energy consumption of various components of the processor [21], [35]. The RAPL interface of most modern Intel processors includes the PP0, PP1 and PKG power domains: PP0 refers to the CPU cores; PP1 refers to the integrated GPU; PKG refers to the entire socket.

2.4. Pixel stealing attacks

Same-origin policy. The same-origin policy is a browser security policy that isolates web pages with different *origins*, where a page's origin is the combination of its scheme (e.g., HTTP or HTTPS), host and port number. This policy helps prevent malicious websites on the internet from interacting with or reading from other websites that the user may visit (e.g., an email provider or company intranet). For example, if a framing web page embeds a cross-origin web page within an iframe, the same-origin policy prevents the framing web page from accessing the contents of that iframe.

SVG filters. While a framing web page may not access the contents of cross-origin iframes, web standards allow the framing web page to apply graphical transformations on top of iframes (and other HTML elements). These transformations can be defined via the `<filter>` CSS element and include dozens of visual effects, from blur and color shift to more complex ones such as 3D lighting. Most of these filters are borrowed from the structured vector graphics (SVG) standard and are commonly referred to as *SVG filters* [36].

Pixel stealing attacks. A *pixel stealing attack* occurs when a web page uses side channels to learn information about pixels it does not have access to programmatically. This type of attack was first presented by Stone [37] (and, concurrently, Kotcher et al. [38]) in 2013. Stone's work demonstrated

1. Specifically, the power budget is computed as the difference between the reactive limit values and the average power consumption [10]

that a framing web page could leak, pixel-by-pixel, a black and white representation of the content rendered within a cross-origin iframe, in violation of the same-origin policy. The attack leveraged a pixel-color-dependent fast path in the Firefox implementation of the `feMorphology` SVG filter. Specifically, it used an SVG filter stack to isolate, binarize, expand, and apply `feMorphology` on each target pixel, which resulted in a different rendering time depending on whether the pixel was black or white. The framing web page could measure this rendering time difference using the `requestAnimationFrame` API (invoked when page rendering completes). Subsequent work demonstrated pixel stealing attacks on Firefox, Chrome, and Safari that used a similar framework but exploited pixel-color-dependent timing variations in floating-point instructions [39], [40].

To mitigate these attacks, browsers adapted their SVG filter implementations to run in constant time [41], [42], [43], [44]. In 2022, Firefox further deployed Total Cookie Protection [45], [46], which partitions cookie storage by the top-level domain being visited. In 2020, Chrome [47] (and, temporarily, Firefox [48]) changed the default value of the `SameSite` attribute for cookies to `Lax`, which prevents cookies from being sent for cross-origin frame requests unless `SameSite` was explicitly set to `None`. Moreover, many modern web sites that contain sensitive data disallow cross-origin framing via the X-Frame-Options HTTP header and the frame-ancestors Content-Security-Policy directive.

Recent work which is closely related to ours "revived" pixel stealing attacks by exploiting pixel-color-dependent power consumption [11], [12] and data compressibility differences [49] in Chrome's constant-time SVG filter implementations. Additionally, recent work from O'Connell et al. shows that on unsupported system configurations, such as devices using outdated or buggy drivers, Chrome resorts to non-constant-time SVG filter implementations that enable pixel stealing attacks via the cache side channel [50]. At the time of writing, all these attacks still work against websites that explicitly set `SameSite=None` and are not protected via the X-Frame-Options header or the frame-ancestors directive. These include, for example, Wikipedia [49].

3. Reverse engineering Thread Director

In this section, we reverse engineer the Thread Director optimization on a modern Intel Meteor Lake processor, revealing the precise conditions influencing Thread Director's hints to the OS. Section 3.1 focuses on the workload classification component, used to determine a running thread's class ID. Section 3.2 focuses on the Thread Director table component, used to determine the performance and energy-efficiency capabilities of different processor cores.

Experimental setup. We run our experiments on two ASRock NUCS BOX mini PCs with the Meteor Lake processors listed in Table 1. Both machines run Windows 11 (version 24H2) which, at the time of writing, is the only OS with Thread Director support. We use the default Windows configuration ("Balanced" power plan). To avoid side effects due to microarchitectural resource contention, we run this section's experiments with Hyper-threading disabled.[2] To read MSRs, we use the WinRing0 driver [51]. To sample the processor power consumption, we use the `MSR_{PKG,PP0,PP1}_ENERGY_STATUS` MSRs. We sample the P-state of a core by reading the `IA32_PERF_STATUS` MSR from that core. To read the Thread Director Table from physical memory, we use the inpoutx64 driver [51].

Table 1. CPUs TESTED IN OUR EXPERIMENTAL SETUPS. BOTH MACHINES ALSO HAVE TWO LP E-CORES (NOT SHOWN ON THE TABLE). FREQUENCY NUMBERS ARE REPORTED IN GHZ.

Model	Cores		Base Freq.		Max Freq.		GPU (X^e)	GPU
	P	E	P	E	P	E	Cores	Max Freq.
125H	4	8	1.2	0.7	4.5	3.6	7	2.2
155H	6	8	1.4	0.9	4.8	3.8	8	2.25

3.1. Workload classification

We start by characterizing the workload classification component of Thread Director. Recall from Section 2.2 that this component assigns a class ID (from 0 to 3) to each running thread depending on that thread's instruction mix. Intel indicates, roughly, the types of instructions that fall within each class ID [31]. Additionally, Intel provides one example workload for each class ID [30].

Methodology. Using Intel's examples as a guide, we construct three workloads. Class ID 0's workload consists of a loop of register-to-register scalar ALU instructions (`xor`, `add`, and `inc`). Class ID 1's workload consists of a loop of register-to-register Advanced Vector Extensions (AVX) instructions (`vfmaddsub132ps`, `vfmaddsub213ps`, and `vfmaddsub231ps`). Class ID 3's workload consists of a loop of `pause` instructions. We do not construct a workload for class ID 2 but note that class 2 instructions are just higher-priority AVX instructions with similar core-to-core IPC ratios to class 1 instructions [31], [52].

We confirm that each workload classifies to the intended class as follows. First, we reset the Thread Director history using the `hreset` instruction [21], which also causes the `IA32_THREAD_FEEDBACK_CHAR` (class ID) MSR to be marked as invalid. Then, we execute the target workload until the class ID MSR is marked as valid. Once valid, we verify the reported class ID against the intended one. Following this approach, we verify that Thread Director consistently classifies our workloads as intended on both P-cores and E-cores. Additionally, we observe that the class ID is updated within 6 to 21 μs of execution in all cases. This is consistent with Intel's claim that Thread Director communicates with the OS on a microsecond level [52].

1. Thread Director communicates the class ID of running threads to the OS on a microsecond level.

2. We verify that our findings also apply with Hyper-threading enabled.

Table 2. CPU POWER CONSUMPTION (PP0) WHEN RUNNING EACH CPU STRESSOR ON ALL P-CORES AND E-CORES (OF OUR 155H PROCESSOR) AND WITH ALL CORES RUNNING AT THEIR BASE FREQUENCY.

Instruction	imul	and	$\texttt{xor}_\texttt{l}$	$\texttt{xor}_\texttt{h}$
PP0 power (W)	7.75	9.02	9.42	10.41

Table 3. GPU POWER CONSUMPTION (PP1) WHEN RUNNING EACH GPU STRESSOR ON OUR 155H PROCESSOR, WITH THE GPU RUNNING AT ITS MAX FREQUENCY. FMUL MAKES PARTIAL USE OF THE GPU.

Instruction	fmul	$\texttt{fma}_\texttt{l}$	$\texttt{fma}_\texttt{h}$
PP1 power (W)	11.83	21.75	27.05

3.2. Thread Director table

We now analyze the table component of Thread Director. Recall from Section 2.2 and Figure 1 that this component consists of a table storing performance and energy efficiency *capabilities* for each CPU core. Intel reports that these capabilities are dynamically updated based on the TDP, operating conditions, and power settings without any user input [31]. Our goal in this section is to analyze how these capabilities depend on the processor's power consumption.

Methodology. We use a series of CPU and GPU *stressor* programs. The CPU stressors consist of n threads, each executing an infinite loop of ALU instructions (all class 0). We vary the specific ALU instructions running inside the loop between imul, and, and xor—which we select because they result in different power consumptions in our experimental setup (cf. Table 2). We additionally create two variants of the xor CPU stressor: a high-power $\texttt{xor}_\texttt{h}$, whose execution has a high Hamming distance and Hamming weight, and low-power $\texttt{xor}_\texttt{l}$, whose execution has a low Hamming distance and Hamming weight.[3]

The GPU stressors consist of OpenCL workloads with kernels executing loops of either floating-point multiplication and addition (fma) or only floating-point multiplication operations (fmul) for approximately 1 minute. In the first stressor, $\texttt{fma}_\texttt{h}$, the kernel performs multiplication and addition operations on random numbers, the work-group size is CL_DEVICE_MAX_WORK_GROUP_SIZE, and the number of work-groups is CL_DEVICE_MAX_COMPUTE_UNITS. The second stressor, $\texttt{fma}_\texttt{l}$, is a variant of $\texttt{fma}_\texttt{h}$ that operates on zero values instead of random values. In the third stressor, fmul, the kernel performs only multiplications on zero values and the number of work-groups is only 4, resulting in a lower power consumption (cf. Table 3).

To avoid side effects due to scheduling, we pin each thread of this section's CPU stressors to a unique CPU core.[4] We run each stressor in two setups: in the first, we disable Turbo Boost, reducing the likelihood that the processor starts throttling [9]; in the second, we leave Turbo Boost enabled. We sample each core's P-state every 10 ms and the Thread Director table every 50 ms.[5] We collect 30,000 P-state and 6,000 Thread Director table samples and use their averages for our analyses. For simplicity, we use the Core Ultra 155H machine (with 6 P-cores) for our discussion, but analogous findings apply to all machines in our experimental setup.

3. The power consumption on Intel CPUs depends on the Hamming distance and Hamming weight of the data being processed [9].

4. We verify that when a thread is pinned to a single core, it runs on that core regardless of Thread Director's hints.

Table 4. THREAD DIRECTOR TABLE'S DEFAULT CAPABILITIES ON OUR 155H PROCESSOR, REPORTED WHEN THE CPU CORES ARE ACTIVE AND THE PROCESSOR IS NOT THROTTLING. "Cn" STANDS FOR "CLASS n".

Core	Performance (Perf)				Energy Efficiency (EE)			
	C0	C1	C2	C3	C0	C1	C2	C3
$\texttt{P}_{0-1}$	67	105	168	43	33	44	51	21
$\texttt{P}_{3-5}$	63	98	157	40	35	46	54	22
$\texttt{E}_{0-3}$	41	41	41	38	124	101	77	113
$\texttt{E}_{4-7}$	41	41	41	38	115	94	71	105
$\texttt{LP}_{0-1}$	0	0	0	0	0	0	0	0

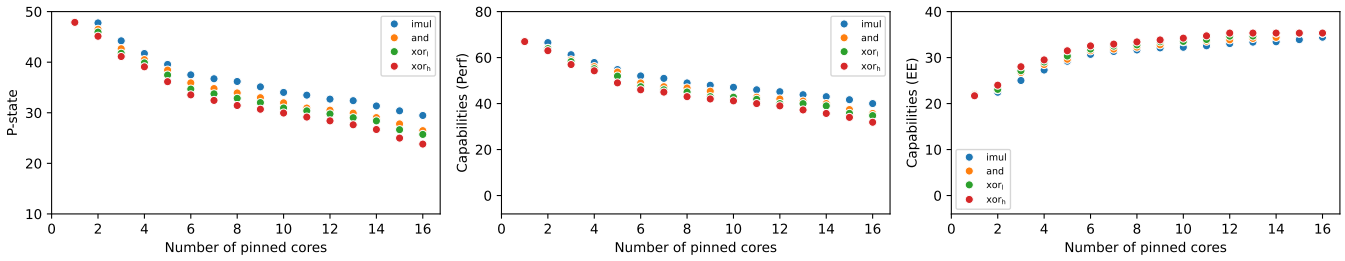
CPU-only workloads. We start by measuring how the Thread Director table capabilities change as a function of the number of active CPU cores and the CPU power consumption. To this end, we run each CPU stressor from Table 2 with a variable number of threads n (with $1 \leq n \leq N$, where N is the total number of cores) while pinning threads in the order of P-cores, E-cores, and LP E-cores.

When we run the above experiment with Turbo Boost disabled, the processor does not throttle and the capabilities in the Thread Director table are always the same regardless of the stressor and the number of threads. We call these capabilities, reported in Table 4, the *default capabilities*. For all class IDs in this configuration, P-cores consistently have higher performance capabilities than E-cores and E-cores consistently have higher energy-efficiency capabilities than P-cores. For each P-core, the class capabilities are ranked such that C2 $>$ C1 $>$ C0 $>$ C3 for both performance and energy efficiency. For each E-core, the class capabilities are ranked such that C2 $=$ C1 $=$ C0 $>$ C3 for performance, and C2 $<$ C1 $<$ C3 $<$ C0 for energy efficiency. Further, the default capabilities of LP E-cores are always set to zero. These observations are consistent with Intel's official description of Thread Director [31] and with their claim that the LP E-cores are intended for "low-utilization, parasitic background tasks" [53], [54]. As we do not experiment with low-utilization tasks, we exclude the LP E-core capabilities from our analyses in the remainder of this paper.

2. When running CPU-only workloads and the processor does not throttle, the Thread Director table always reports the *default capabilities* regardless of the power consumption and the number of threads.

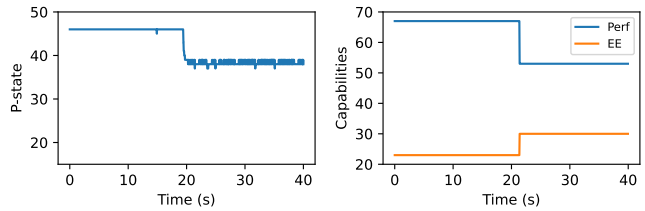
Next, we run our experiments with Turbo Boost enabled. Here, we observe that the Thread Director table starts with the default capabilities but dynamically updates them after

5. We empirically observe that the Thread Director table updates at a coarse granularity, on the order of hundreds of milliseconds.



(a) P-state vs Number of active cores (b) Perf capabilities vs Number of active cores (c) EE capabilities vs Number of active cores

Figure 2. Average P-state, performance capabilities, and energy-efficiency capabilities of the active P-cores when running the ALU-only CPU stressors with a variable number of threads (with each thread pinned to a unique CPU core in the order of P-cores, E-cores, and LP E-cores).



(a) P_0 P-state (b) P_0 capabilities

Figure 3. P-state and class 0 capabilities of P_0 when running 6 `and` CPU stressors on the P-cores of our 155H processor. The P-state and performance capabilities drop, while the energy efficiency capabilities grow shortly after the processor starts throttling.

the processor begins throttling. Consider, for example, the case when we run the `and` CPU stressor with $n = 6$ threads pinned to cores P_{0-5}. Figure 3a shows the P-state of core P_0 during one run of the experiment. The P-state starts at 46, corresponding to a 4.6 GHz frequency. This P-state is sustained for about 20 s, during which the Thread Director table reports the default capabilities. Then, the processor begins throttling, reducing the P-state to remain below safe power and thermal limits. Figure 3b shows the performance and energy-efficiency capabilities of P_0 during the same time period. Here, the capabilities start at their default values. However, after the P-state drops, the performance capability decreases and the energy-efficiency capability increases. That is, after the processor begins throttling, the reported capabilities change to reflect variations in the P-state, which in turn are influenced by the processor's power consumption [9]. This behavior is consistent across cores and is supported by Intel manuals, which state that Thread Director's feedback is "dynamic" and "based on power/thermal limits" [30]. We refer to the capabilities in this configuration as *dynamic capabilities*.

Figure 2 shows the results for other stressors and numbers of threads. In particular, Figure 2a reports the average P-state of all active P-cores, and Figures 2b and 2c report the average P-core capabilities during each experiment. Here, we observe that the performance capabilities are lower and the energy-efficiency capabilities are higher when the P-state is lower, confirming that these capabilities are dynamically updated to reflect differences in the processor's P-state (and, hence, power consumption [9]). We observe the same behavior in the E-cores: the only difference is that the E-core energy-efficiency capabilities also shift by a fixed amount depending on the active states of the two E-core clusters.

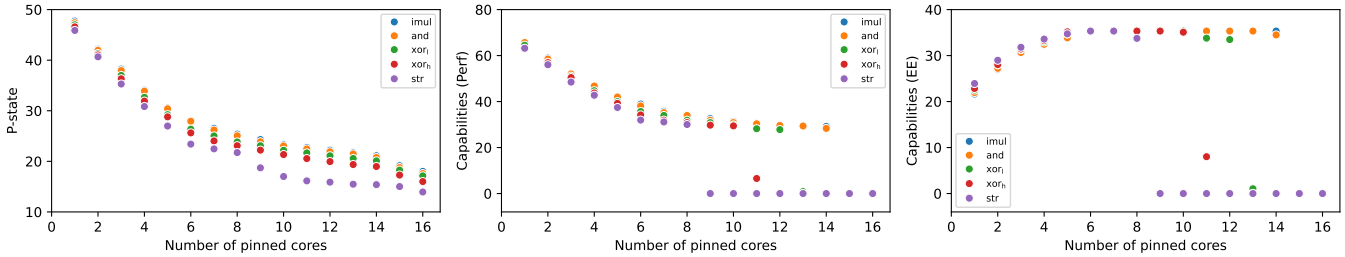
> 3. When running CPU-only workloads and the processor throttles, the Thread Director table reports *dynamic capabilities* which reflect changes to the CPU power consumption and P-states. A higher power consumption results in lower performance capabilities and higher energy-efficiency capabilities on all cores.

Having established that the Thread Director table is dynamically updated when the processor throttles, we now investigate whether the dynamically updated capabilities result in changes to the class or core type rankings compared to the default capabilities.[6] To this end, we analyze all the Thread Director table samples collected during the experiments of Figure 2. We make two observations. First, for each CPU core, the class capabilities are always ranked the same way as in the default capabilities. For example, for any given P-core, the dynamic class capabilities are still ranked such that C2 > C1 > C0 > C3 for both performance and energy efficiency. Second, when running CPU stressors with n threads, the core types of the most performant and most efficient n cores do not change between default and dynamic capabilities. For example, when running $n = 4$ stressor threads, the 4 cores with the highest performance capabilities are still all P-cores and the 4 cores with the highest energy-efficiency capabilities are still all E-cores.

> 4. When running an n-thread CPU workload, the core types of the most performant and most efficient n cores are the same between the default and the dynamic capabilities. Also, the ranking between each core's class IDs is always the same as in the default capabilities.

Intel manuals mention that Thread Director's feedback can also come in the form of "idling hints", when "power

6. Recall from Section 2.2 that when it is time for the OS to schedule a thread, the scheduler can use the Thread Director table to determine the most performant and efficient cores for that thread. These cores depend on the ranking of capabilities of the thread's class between different cores.



(a) P-state vs Number of active cores (b) Perf capabilities vs Number of active cores (c) EE capabilities vs Number of active cores

Figure 4. Average P-state, performance capabilities, and energy-efficiency capabilities of the active P-cores when running the memory-intensive CPU stressors with a variable number of threads (with each thread pinned to a unique CPU core in the order of P-cores, E-cores, and LP E-cores).

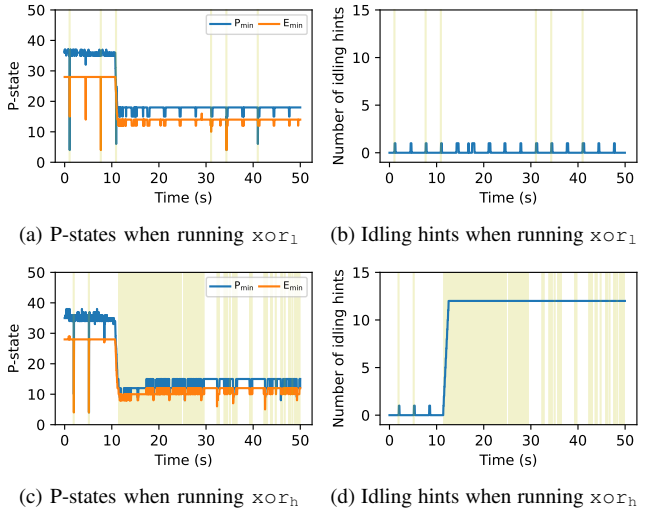
Table 5. CPU POWER CONSUMPTION (PP0) WHEN RUNNING THE MEMORY-INTENSIVE CPU STRESSORS ON ALL P- AND E-CORES (OF OUR 155H PROCESSOR), WITH CORES RUNNING AT BASE FREQUENCY.

Instruction	`imul`	`and`	$\texttt{xor}_\texttt{l}$	$\texttt{xor}_\texttt{h}$	`str`
PP0 power (W)	12.70	13.18	13.49	14.14	15.21

and thermal are constrained" [30]. We hypothesize that these hints correspond to setting certain cores' capabilities to zero, indicating "a recommendation to the OS to not schedule any software threads" on those cores [21]. However, we do not observe any such cases in Figure 2's experiments. Below, we investigate two methods to trigger said idling hints.

First, we repeat the experiments of Figure 2, but this time we increase the power consumption of our CPU stressors by introducing high memory utilization. Specifically, we modify each thread to allocate a 256 KB array and execute loads from this array in addition to the stressor's ALU instructions. We also include an additional `str` CPU stressor, which performs memory stores to a 4 MB array in addition to the loads. Table 5 shows the power consumption of these memory-intensive CPU stressors, and Figure 4 shows the results. The initial behavior is similar to that of Figure 2: as the power consumption increases, the core capabilities become dynamic, reflecting changes to the P-states. However, as the power consumption continues to grow (e.g., when running the memory-intensive $\texttt{xor}_\texttt{h}$ stressor with $n \geq 11$ threads), the P-core capabilities start dropping more drastically until they are all set to zero. We observe the same behavior in the E-cores, with the exception of the last two cores ($\texttt{E}_{6\text{-}7}$), whose capabilities remain non-zero even when all other core capabilities are zero. Following Intel's terminology [30], we refer to zero core capabilities as "idling hints".

We then analyze the samples collected during Figure 4's experiments. Here, we observe that idling hints consistently appear whenever the following two conditions are met: (i) the processor is throttling and (ii) the P-state of at least one core falls below a certain value. On our 155H processor, this value appears to be 15 for P-cores and 12 for E-cores, respectively, and the P-states of $\texttt{P}_1$ and $\texttt{E}_{4\text{-}7}$ are consistently the first ones to drop below (and last ones to return above)



(a) P-states when running $\texttt{xor}_\texttt{l}$ (b) Idling hints when running $\texttt{xor}_\texttt{l}$

(c) P-states when running $\texttt{xor}_\texttt{h}$ (d) Idling hints when running $\texttt{xor}_\texttt{h}$

Figure 5. P-states and number of cores with idling hints when running the memory-intensive $\texttt{xor}_\texttt{l}$ or $\texttt{xor}_\texttt{h}$ CPU stressors with $n = 12$ threads, each thread pinned to a unique CPU core. After the processor starts throttling, idling hints occur only in the $\texttt{xor}_\texttt{h}$ stressor (Figure 5d), where the minimum P-core P-state ($\texttt{P}_\texttt{min}$) or the minimum E-core P-state ($\texttt{E}_\texttt{min}$) fall below (yellow regions) and remain at or below 15 and 12, respectively.

these values.[7] When the processor throttles and the P-state of at least one core falls below these values, idling hints begin to appear starting from $\texttt{P}_0$ and progressing core by core until reaching $\texttt{E}_5$. Conversely, when the processor stops throttling or the P-state of all cores goes above these values, idling hints begin to disappear in the reverse order—starting from $\texttt{E}_5$ and progressing core by core until reaching $\texttt{P}_0$. When the processor throttles and the P-states of at least one core is exactly at these values, idling hints do not change.

For example, Figure 5 shows the P-states and the number of cores with idling hints during one run of the $\texttt{xor}_\texttt{l}$ and $\texttt{xor}_\texttt{h}$ stressors with $n = 12$ threads. Before the processor starts throttling, no cores have idling hints. After the processor starts throttling, idling hints occur only in the $\texttt{xor}_\texttt{h}$ stressor, where the P-state of at least one core falls below the aforementioned values (yellow regions) and remains at or below these values for the duration of the experiment.

7. Each core can have its P-state adjusted independently (cf. Section 2.3), but the 4 cores in each E-core cluster always run at the same P-state.

To corroborate our findings, we run additional experiments designed to satisfy the above conditions and trigger idling hints. To satisfy the first condition (P-states below certain values), we manually lower the maximum P-state of P-cores to 9 and E-cores to 8. To satisfy the second condition (processor throttling) while operating at reduced frequency levels, we run our workloads with a reduced processor's power limit (manually configured via the `MSR_PKG_POWER_LIMIT` MSR). In this setup, we observe that idling hints *always* occur whenever the processor throttles.

These experiments are consistent with Intel's claim that when the power budget is constrained, "an E-core may become the most performant core or operating on a limited set of cores may provide best system performance" [55].

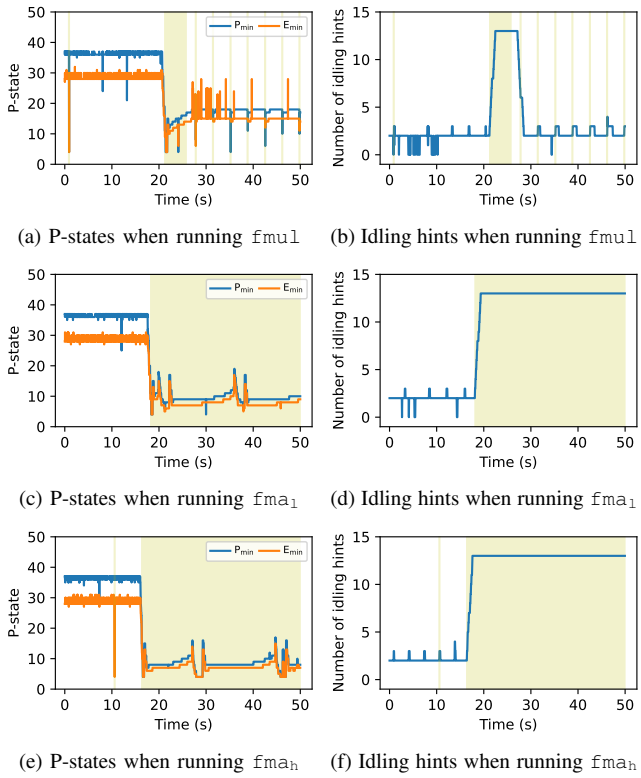
> 5. When running CPU-only workloads, the processor throttles, and the P-state of at least one core falls below a certain value, the Thread Director table gives *idling hints*, which consist of cores with zero capabilities. When these conditions are met, idling hints gradually appear in core order on all cores except for $\texttt{E}_{6\text{-}7}$. When some cores have idling hints, the ranking of performance capabilities between P-cores and E-cores changes compared to the default capabilities.

Activating the GPU. We now investigate a second method to trigger idling hints by running stressors on the integrated GPU. By doing so, we aim to reduce the available power budget of the CPU and induce idling hints without the need for high memory utilization or reduced power limits.

To study how the Thread Director table changes when both CPU and GPU are active, we repeat the experiments from Figure 2 while simultaneously running the GPU stressors from Table 3. We make the following observations. First, the first two P-cores ($\texttt{P}_{0\text{-}1}$) always show idling hints when the GPU is active and the number of CPU stressor threads is $n \leq 8$, regardless of the processor's power consumption. Intel patents and videos suggests that this behavior may be intended to redirect the CPU power budget to the GPU to boost GPU performance [55], [56], [57].

> 6. When the GPU is active and the number of active CPU cores is $n \leq 8$, Thread Director always gives idling hints on the first two P-cores $\texttt{P}_{0\text{-}1}$.

Second, when the GPU is active, the processor throttles, and the P-state of at least one core falls below a certain value, idling hints begin to appear on other cores too. On our 155H processor, this value appears to be 17 for P-cores and 14 for E-cores, respectively. When $n \leq 8$, these idling hints start from $\texttt{P}_2$ and progress core by core until reaching $\texttt{E}_6$. When $n > 8$, they start from $\texttt{P}_0$ and progress until reaching $\texttt{E}_5$. Once the processor stops throttling or the P-state of all cores goes above these values, idling hints begin to disappear in the reverse order. Finally, when the processor throttles and the P-states of at least one core is exactly at these values, idling hints do not change.



(a) P-states when running `fmul` (b) Idling hints when running `fmul`

(c) P-states when running $\texttt{fma}_\texttt{l}$ (d) Idling hints when running $\texttt{fma}_\texttt{l}$

(e) P-states when running $\texttt{fma}_\texttt{h}$ (f) Idling hints when running $\texttt{fma}_\texttt{h}$

Figure 6. P-states and number of idling hints when the GPU runs the `fmul`, $\texttt{fma}_\texttt{l}$, and $\texttt{fma}_\texttt{h}$ GPU stressors and the CPU runs the $\texttt{xor}_\texttt{h}$ CPU stressor with $n = 8$ threads (pinned to $\texttt{P}_{0\text{-}5}$, $\texttt{E}_0$, and $\texttt{E}_4$). In all cases, before the processor throttles, there are 2 idling hints (due to the GPU being active and $n \leq 8$). After the processor starts throttling, idling hints appear on all cores (except for $\texttt{E}_7$). These idling hints are due to the minimum P-core P-state ($\texttt{P}_{\texttt{min}}$) or the minimum E-core P-state ($\texttt{E}_{\texttt{min}}$) falling below (yellow regions) and remaining at or below 17 and 14, respectively. This occurs consistently when running the $\texttt{fma}_\texttt{h}$/$\texttt{fma}_\texttt{l}$ stressors, and only temporarily when running the lower-power `fmul` stressor.

For example, Figure 6 shows the P-states and the number idling hints when the GPU runs the `fmul`, $\texttt{fma}_\texttt{l}$, and $\texttt{fma}_\texttt{h}$ GPU stressors and the CPU runs the $\texttt{xor}_\texttt{h}$ stressor with $n = 8$ threads pinned to $\texttt{P}_{0\text{-}5}$, $\texttt{E}_0$, and $\texttt{E}_4$. In all cases, before the processor starts throttling, only $\texttt{P}_{0\text{-}1}$ have idling hints; these are due to the GPU being active and $n \leq 8$. After the processor starts throttling, idling hints start occurring on other cores too (except for $\texttt{E}_7$); these are due to the P-state of at least one core falling below the aforementioned values (yellow regions) and subsequently remaining at or below these values This occurs consistently when running the $\texttt{fma}_\texttt{h}$ and $\texttt{fma}_\texttt{l}$ stressors, and temporarily when running the lower-power `fmul` stressor. The only difference between the $\texttt{fma}_\texttt{h}$ and $\texttt{fma}_\texttt{l}$ stressors is that idling hints occur earlier when running $\texttt{fma}_\texttt{h}$ due to its higher power consumption. We observe analogous results with other CPU stressors. Moreover, when we manually lower the maximum P-state of P-cores to 9 and E-cores to 8, we observe that idling hints always occur whenever the processor throttles.[8]

8. In this case, the GPU power consumption is high enough to induce throttling (and thus idling hints) despite the reduced frequency levels.

7. When the GPU is active, the processor throttles, and the P-state of at least one core falls below a certain value, Thread Director gives idling hints for all cores except for E_7 (when the number of active CPU cores is $n \leq 8$) or E_{6-7} (when $n > 8$). The higher the GPU power consumption, the earlier the processor throttles and Thread Director gives these idling hints.

Summary. Putting all this together, the Thread Director table can show three types of capabilities:

1) *Default capabilities:* These are fixed (cf. Table 4), appear when the processor does not throttle, and do not depend on the processor's power consumption.
2) *Dynamic capabilities:* These appear when the processor throttles; they depend on the processor's power consumption but do not affect the P-core to E-core capability ranking compared to the default capabilities.
3) *Idling hints:* These occur in two cases:
   a) Consistently on P_{0-1} when the GPU is active and there are $n \leq 8$ active CPU cores.
   b) Core-by-core on all cores except for E_{6-7} or E_7 (depending on the CPU and GPU active states) when the processor throttles and the P-state of at least one core falls below a certain value.

   In case (b), idling hints depend on the processor's power consumption. In both case (a) and case (b), idling hints change the P-core to E-core capability ranking compared to the default capabilities.

4. Analyzing the impact of Thread Director on the Windows scheduler

In this section, we analyze how the Windows OS scheduler uses Thread Director's hints. Section 4.1 analyzes how scheduling varies as a function of Thread Director's class IDs. Section 4.2 analyzes how scheduling varies as a function of the capabilities stored in the Thread Director table.

Experimental setup. We use the same experimental setup as Section 3. Additionally, we sample the core where a thread is running on by calling the `GetCurrentProcessorNumber` Windows API from said thread.

4.1. Workload classification

We now analyze how scheduling on Windows varies as a function of class IDs. In particular, we demonstrate that a process can "game" the Windows scheduler into making different decisions solely as a function of its threads' class IDs. As we show in Section 5, this behavior can be exploited to build a reliable cross-core covert channel.

Methodology. We aim to understand whether Windows gives different priorities on P-cores to threads based on their class IDs. To this end, we use the workloads from Section 3.1 to construct three *receiver* programs. Each receiver runs multiple threads, each executing a loop of instructions of a specified class ID and sampling the core where it runs every 10 ms. We run each pairwise combination of receivers simultaneously and vary the number of threads of each receiver. For all combinations of receivers and thread counts, we calculate the total percentage of time during which each receiver runs on P-cores (across all its threads). To isolate the class ID-dependent scheduling behavior and avoid side effects due to throttling (analyzed in Section 4.2), we disable Turbo Boost. Figure 7 shows the results. We discuss the results for our Core Ultra 155H machine, but the results apply to both our machines.

Observations. First, when the total number of threads across both receivers is equal to or lower than the number of P-cores, all threads run on the P-cores. This is consistent with our observation from Section 3.2: since P-cores consistently have higher performance capabilities than E-cores, the scheduler prioritizes them for threads of all class IDs.

Second, when two receivers with the same class ID run simultaneously, the scheduler gives their threads equal priority on the P-cores, as shown in the top-left to bottom-right diagonal of Figure 7. In this case, if the number of threads is greater than the number of P-cores, the scheduler uses its default round-robin scheduling policy, where each thread gets equal time slices on the P-cores [22].

Third, when two receivers with different class IDs run simultaneously, the scheduler gives priority on P-cores to the receiver whose class ID has higher P-core performance capabilities. Thus, a class 1 receiver always gets priority on the P-cores over a class 0 or 3 receiver, and a class 0 receiver always gets priority on the P-cores over a class 3 receiver. This ranking matches that of the default performance capabilities, where C1 $>$ C0 $>$ C3 (cf. Section 3.2).

8. When threads with different class IDs run simultaneously, Windows prioritizes those with higher P-core performance capabilities on P-cores.

Security implications. The above mechanism deviates from Windows' [22] (and other commonly used schedulers' [58], [59]) stated goal of treating same-scheduling priority threads as equal. This observation comes with several security implications. First, by monitoring its P-core usage, a thread may be able to infer the class ID of other threads running on the system, even when those threads never share a core. For example, a class 0 thread that never gets scheduled on the P-cores may infer that other threads are executing class 1 instructions. In Section 5, we exploit this behavior to mount a covert channel. Second, although we do not explore these scenarios in this paper, a malicious program may attempt to get priority on P-cores by adding class 1 instructions to its own threads. This may be exploited, for example, by certain types of software (e.g., videogames, cryptojacking malware) to boost their performance at the cost of other programs or by a VM [60], [61] to bias scheduling decisions at the cost of other VMs. Finally, the ability to manipulate

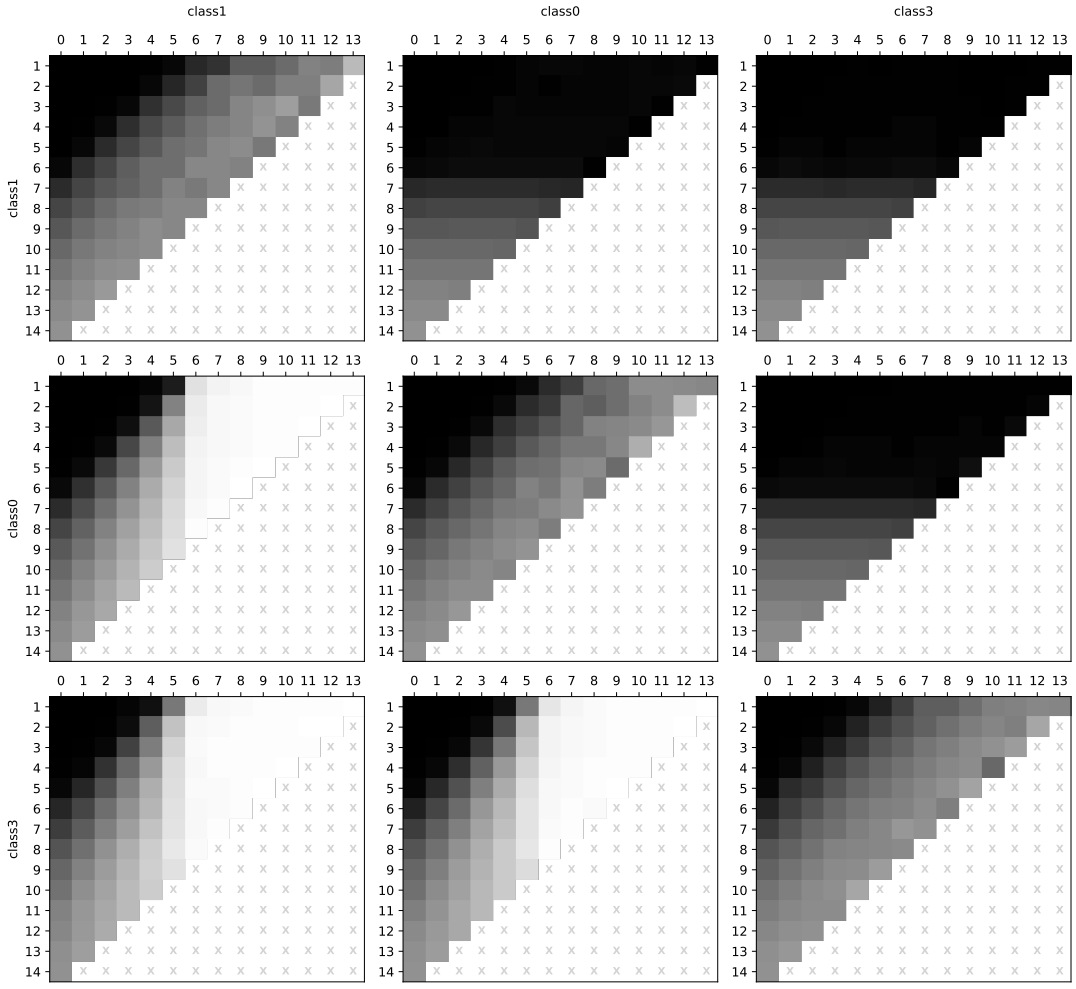


Figure 7. Class scheduling heatmap on our 155H machine (with 6 P-cores). The y-axis label represents the class ID of the receiver whose P-core usage we report in that row. The x-axis label indicates the class ID of the receiver running simultaneously with the y-axis receiver. Tick marks within each subplot indicate the number of threads of the respective receiver. Cells marked with an "×" indicate invalid configurations exceeding the number of cores. Darker cells indicate higher P-core usage, meaning that the y-axis receiver threads spent more time running on P-cores in that configuration: a black cell means that the y-axis receiver spent 100% of its time on P-cores, whereas a white cell means that the y-axis receiver spent 0% of its time on P-cores.

the scheduler and could be abused to facilitate co-location between programs and slow down other programs. These capabilities have been shown to facilitate microarchitectural side-channel attacks [62], [63], [64].

4.2. Thread Director table

We now analyze how scheduling on Windows 11 varies as a function of varying Thread Director table capabilities. In particular, we demonstrate that when Thread Director gives idling hints on specific cores, the Windows scheduler does not schedule any threads on those cores. As we show, this results in a power consumption-dependent scheduling behavior where the average number of active cores varies depending on the processor's power consumption. This observation undergirds both side-channel attacks of Section 6.

Methodology. In this section, we rely on a new *core sampling CPU stressor*. This stressor consists of n threads, each executing a loop of instructions from Section 3.2's CPU stressors and sampling the current core where it is running every 5 ms. Unlike in Section 3.2, we do not pin the threads of this section's CPU stressors to any specific core, allowing the scheduler to place threads wherever it sees fit.

Dynamic capabilities. We start by analyzing whether the scheduling behavior changes when the Thread Director table reports dynamic capabilities compared to the default capabilities. Recall from Section 3.2 that dynamic capabilities

appear when the processor throttles. To understand if dynamic capabilities affect scheduling, we run the ALU-only core sampling CPU stressors with a variable number of threads n (with $1 \leq n \leq N$, where N is the number of cores).[9] For each experiment, we measure the percentage of samples the CPU stressor's threads spend on P-cores (i.e., the P-core usage). In all cases, we observe no statistically significant difference in the P-core usage when the table reports the default capabilities (the processor does not throttle) and when it reports dynamic capabilities (the processor throttles). This is consistent with our observation from Section 3.2 that the core types of the n most performant and most efficient cores when running an n-thread workload are the same between the default and the dynamic capabilities.

> 9. The P-core usage of unpinned CPU-only workloads is the same whether the Thread Director table reports dynamic capabilities or the default capabilities.

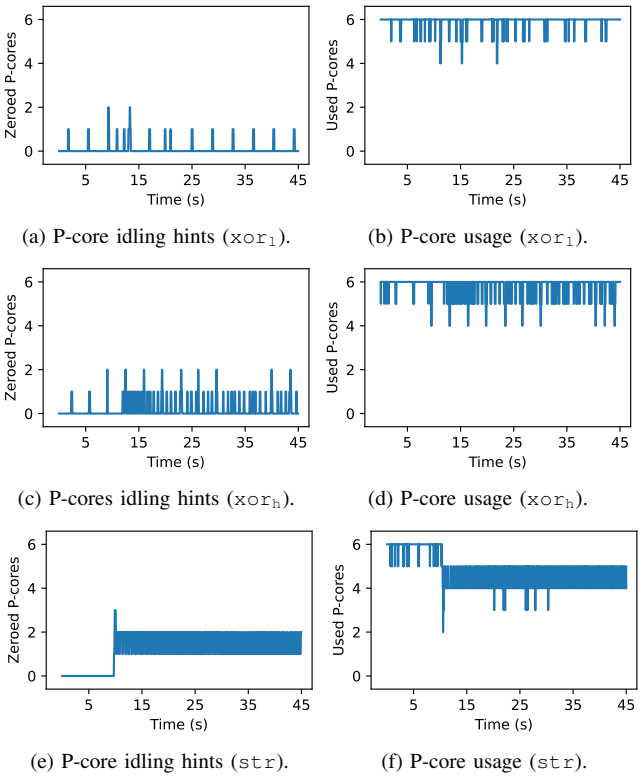
Idling hints. We now analyze whether the scheduling behavior changes when the Thread Director table gives idling hints. Recall from Section 3.2 that idling hints occur either (i) when the GPU is active and there are $n \leq 8$ active CPU cores, or (ii) when the processor is throttling and the P-state of at least one core falls below a certain value.

We start by investigating how the idling hints caused by our memory-intensive CPU stressors affect scheduling. To this end, we run the memory-intensive core sampling CPU stressors with a variable number of threads n while monitoring the Thread Director table and P-states. As in Section 3.2, we observe that idling hints begin to appear as soon as the processor throttles and the P-state of at least one core falls below 15 (for P-cores) or 12 (for E-cores). However, unlike in Section 3.2, idling hints never reach $\texttt{E}_5$. This is because, as soon as idling hints start appearing, the OS stops scheduling threads on the affected cores, causing them to go idle. These idle cores reduce the processor's power consumption, allowing the P-states to rise. As a result, the idling hints begin to disappear, and the idle cores begin to wake up. This, in turn, causes the power consumption to grow and the P-states to drop again, triggering new idling hints. As this cycle repeats, *the number of active cores oscillates and depends on the power consumption*.

> 10. When Thread Director gives an idling hint on a core (and no threads are pinned exclusively to that core), Windows 11 does not schedule threads on that core.

Figure 8 shows example runs of the $\texttt{xor}_\texttt{l}$, $\texttt{xor}_\texttt{h}$, and `str` memory-intensive core sampling CPU stressors with $n = 14$ threads. When running $\texttt{xor}_\texttt{l}$ and the processor throttles, the P-states never fall below the aforementioned values. As a result, no idling hints occur, and all cores remain active. However, when running $\texttt{xor}_\texttt{h}$ or `str` and the processor throttles, the minimum P-core P-state oscillates between 12, 15, and 18 and the minimum E-core P-state oscillates between 10, 12, and 14. This results in the number of idling hints oscillating between 0 and 1 in the $\texttt{xor}_\texttt{h}$ case or 1 and 2 (in the `str` case). This is directly reflected in the number of active P-cores, which oscillates between 5 and 6 or between 4 and 5, respectively.

9. We confirm that when running these stressors, the P-states are high enough for idling hints not to occur, similarly to what we see in Figure 2.



(a) P-core idling hints ($\texttt{xor}_\texttt{l}$). (b) P-core usage ($\texttt{xor}_\texttt{l}$).

(c) P-cores idling hints ($\texttt{xor}_\texttt{h}$). (d) P-core usage ($\texttt{xor}_\texttt{h}$).

(e) P-core idling hints (`str`). (f) P-core usage (`str`).

Figure 8. Number of idling hints on P-cores and P-core usage when running the memory-intensive $\texttt{xor}_\texttt{l}$, $\texttt{xor}_\texttt{h}$, and `str` core sampling CPU stressors with $n = 14$. When the number of idling hints is higher, the P-core usage is lower, confirming that Windows does not schedule threads on cores with idling hints. Additionally, the number of idling hints (and, hence, idle cores) varies depending on the processor's power consumption.

The above result demonstrates that the average number of active P-cores decreases when the processor's power consumption grows. To corroborate this finding, we re-run Figure 8's experiments with two additional stressor threads pinned to the LP E-cores ($\texttt{LP}_{\texttt{0-1}}$), which are otherwise unused. Here, we observe results similar to Figure 8 but with a larger number of idle cores during throttling. Specifically, the number of active P-cores oscillates mostly between 1 and 2 when running the $\texttt{xor}_\texttt{l}$ and the $\texttt{xor}_\texttt{h}$ stressors, and between 2 and 3 when running the `str` stressor.

> 11. When running unpinned CPU-only workloads, the processor throttles, and the P-state of at least one core falls below a certain value, the number of active cores varies as a function of the processor's power consumption. Specifically, a higher processor's power consumption results in a lower number of active cores.

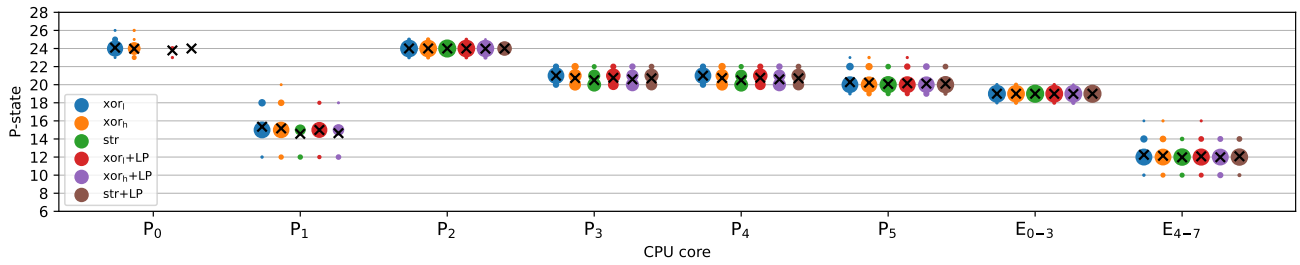


Figure 9. Distribution of P-states when running the memory-intensive CPU stressors with $n = 14$ threads on our 155H processor. The plot includes the results when running each stressor with two additional threads pinned to $\texttt{LP}_{0\text{-}1}$. Circle size represents how frequently each core (x-axis) runs at each P-state (y-axis). For example, $\texttt{E}_{4\text{-}7}$ has a bigger circle at P-state 12 because, most of the time, it runs at P-state 12. The mean P-states for each workload are marked with an "×". With higher-power workloads (e.g., `str`), some cores have smaller (or no) circles due to power-dependent idling hints. However, the P-states of the active cores consistently oscillate between the same values, regardless of the processor's power consumption.

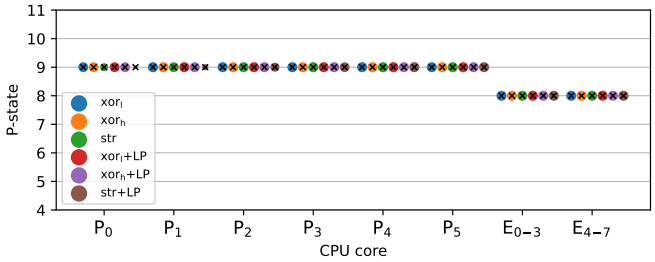


Figure 10. Same as Figure 9, but with the maximum P-state capped at 9 for P-cores and 8 for E-cores and a reduced processor power limit of 20 W.

Figure 9 shows P-state samples collected during the above experiments. Interestingly, we observe that when idling hints occur, the P-states of the active cores consistently oscillate between the same values, regardless of the processor's power consumption. For example, the P-state of $\texttt{P}_1$ oscillates between 12, 15, and 18, spending most of the time at 15. Similarly, the P-state of $\texttt{E}_{4\text{-}7}$ oscillates between 10, 12, and 14, spending most of the time at 12. This behavior suggests that when the processor is throttling and at least one CPU core is unable to run at or above certain P-states (in our case, 15 for P-cores and 12 for E-cores), Meteor Lake's power management algorithm prefers reducing the number of active cores over further lowering the P-states. In these cases, *adjusting the number of active cores—not frequency scaling—appears to be the primary mechanism used to maintain safe operating conditions*. As a result, the number of active cores depends on the power consumption *despite the absence of frequency side-channel leakage*. The P-states are only further reduced if we force the OS to ignore idling hints by manually pinning threads to enough cores (as we do in Figure 4), or if the power budget becomes so constrained that core idling alone is insufficient.

To corroborate this finding, we re-run Figure 8's experiments while manually setting the maximum P-states of P-cores to 9 and E-cores to 8 and reducing the processor power limit to 20 W. In this setup, we still observe that the average number of idle cores is higher when the power consumption is higher. However, as shown in Figure 10, the P-states of the active cores remain constant regardless of the power consumption. This demonstrates that the number of active cores can depend on the power consumption even when the CPU frequency is constant.[10]

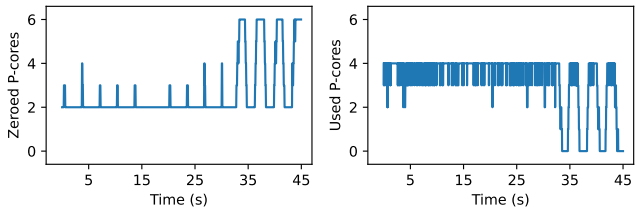
> 12. Meteor Lake's power management algorithm prefers reducing the number of active cores over lowering the CPU frequency below certain values. Once the CPU cores reach these values, the number of active cores can depend on the processor's power consumption even in the absence of frequency side-channel leakage.

Finally, we investigate how the idling hints caused by our GPU stressors affect scheduling. To this end, we execute the CPU core sampling stressors (with n threads) while simultaneously running the GPU stressors from Section 3.2 and monitoring the Thread Director table and P-states. As in Section 3.2, we observe that the first two P-cores ($\texttt{P}_{0\text{-}1}$) always show idling hints when the GPU is active and $n \leq 8$, regardless of the processor's power consumption. This results in 2 P-cores always being idle. Moreover, when the processor starts throttling and the P-state of at least one core falls below 17 (for P-cores) or 14 (for E-cores), idling hints start occurring on other cores too, except for $\texttt{E}_7$ (when $n \leq 8$) or $\texttt{E}_{6\text{-}7}$ (when $n > 8$). As in the CPU-only experiments, when idling hints appear, the OS stops scheduling threads on the affected cores, causing them to go idle. However, due to the high GPU power consumption, idling some CPU cores does not help reduce the processor's power consumption enough for the P-states to return above the aforementioned values. Hence, in our experiments, idling hints always reach $\texttt{E}_6$ (when $n \leq 8$) or $\texttt{E}_5$ (when $n > 8$).[11]

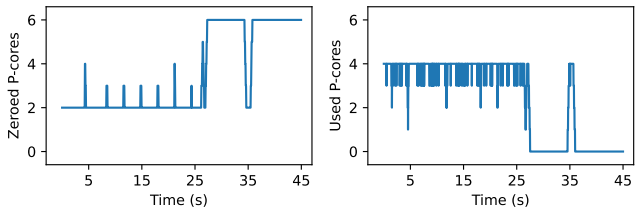
Figure 11 shows the results when the GPU stressors run alongside $n = 6$ $\texttt{xor}_\texttt{h}$ core sampling threads. In all cases, before the processor starts throttling, $\texttt{P}_{0\text{-}1}$ are idle and the other P-cores are active. This is due to the GPU being active and $n \leq 8$. After the processor starts throttling, all cores

10. If we reduce the power limit too much (e.g., to 15 W), reducing the number of active cores is not enough and frequency scaling is used too.

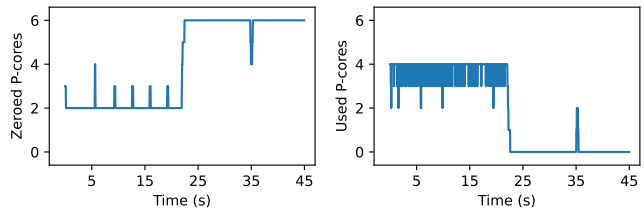
11. We hypothesize that, for lower-power GPU stressors, *the number of* active cores may also depend on the processor's power consumption. However, even with many idle cores, the GPU stressors we use are too power-hungry for the P-states to return above the aforementioned values.



(a) P-core idling hints (`fmul`). (b) P-core usage (`fmul`).



(c) P-core idling hints ($\texttt{fma}_\texttt{l}$). (d) P-core usage ($\texttt{fma}_\texttt{l}$).



(e) P-core idling hints ($\texttt{fma}_\texttt{h}$). (f) P-core usage ($\texttt{fma}_\texttt{h}$).

Figure 11. Number of idling hints on P-cores and P-core usage when running the `fmul`, $\texttt{fma}_\texttt{l}$, and $\texttt{fma}_\texttt{h}$ GPU stressors and the $\texttt{xor}_\texttt{h}$ core sampling CPU stressor with $n = 6$. All CPU threads are scheduled to $\texttt{E}_7$ and the P-core usage goes to zero as soon as the package starts throttling, which happens earlier for GPU stressors using higher power. When running `fmul`, idling hints oscillate since the scheduler listens to the idling hints, causing the processor to periodically (and temporarily) stop throttling.

except for $\texttt{E}_7$ become idle. This is due to the P-states falling below the aforementioned values, which occurs consistently in the $\texttt{fma}_\texttt{h}$ and $\texttt{fma}_\texttt{l}$ case (except for a brief peak around second 35 in the trace) and periodically in the `fmul` case. As in Section 3.2, the only difference between the $\texttt{fma}_\texttt{h}$ and $\texttt{fma}_\texttt{l}$ stressors is that, due to its higher power consumption, $\texttt{fma}_\texttt{h}$ causes all P-cores to become idle earlier.

13. When the GPU is active, the processor throttles, and the P-state of at least one core falls below a certain value, all cores become idle except for $\texttt{E}_7$ (when the number of active CPU cores is $n \leq 8$) or $\texttt{E}_{6\text{-}7}$ (when $n > 8$).

5. Covert channel

We now demonstrate how to use the findings from Section 4.1 to mount a reliable cross-core covert channel which does not require the sharing of any microarchitectural structure nor the use of any (explicit or implicit) timer.

Channel setup. Like prior work, our covert channel makes use of a *receiver* and a *sender*. The receiver supports three configurations (one per class ID) corresponding to the three

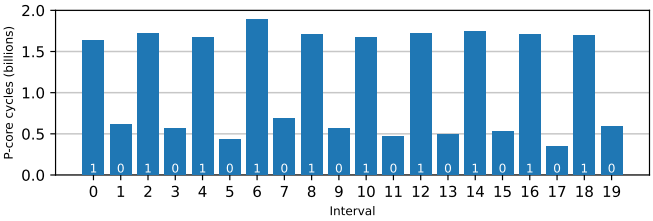


Figure 12. Total time the receiver threads spend on P-cores when the receiver runs class 0 instructions, the sender transmits a sequence of '0's and '1's by alternating class 1 and class 3 instructions, both receiver and sender use 6 threads, and the transmission interval is 375 M cycles. When the sender transmits a '1', the receiver threads spend significantly more time on the P-cores than when the sender transmits a '0'.

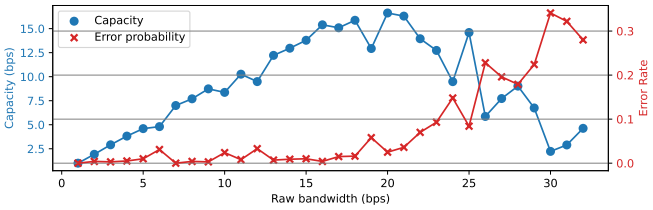
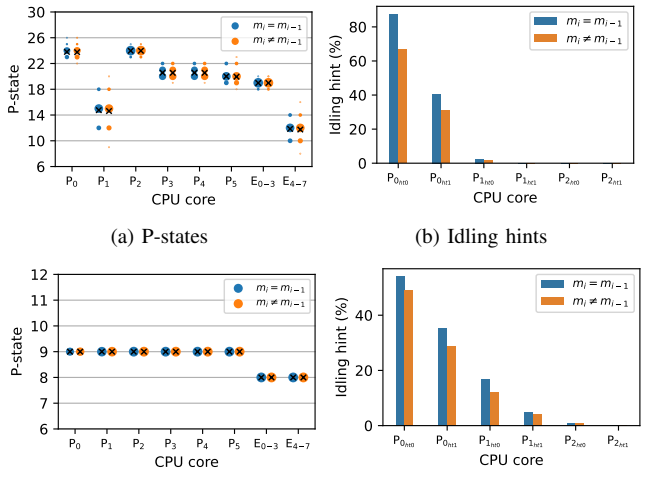


Figure 13. Raw bandwidth, error rate, and channel capacity using Figure 12's configuration with a variable transmission interval.

receivers used in Section 4.1. The sender is a variant of the receiver that can execute instructions of different class IDs depending on the value of the bit being transmitted. For simplicity, we build our sender to only transmit one bit at a time. To transmit a '0', the sender executes a loop of instructions whose class ID has equal or higher priority on P-cores compared to the receiver's instructions. To transmit a '1', the sender executes a loop of instructions whose class ID has lower or equal priority on P-cores compared to the receiver's instructions. The receiver infers each bit's value based on its threads' P-core usage during each interval.

Evaluation. We create a proof-of-concept (POC) implementation of our covert channel, where the sender and the receiver agree on a fixed bit transmission interval and synchronize using the timestamp counter. Consider an example where the receiver runs 6 threads of instructions with class ID 0, and the sender runs 6 threads of instructions with class ID 1 or 3 to transmit a '0' or '1' bit, respectively. Figure 12 shows the P-core usage measured by the receiver when the sender transmits an alternating sequence of bits with an interval of 375 M cycles (equivalent to a raw bandwidth of 8 bps). Every other interval contains high P-core usage measurements, which occur when the sender executes class 3 instructions. These intervals are decoded as '1' bits.

We evaluate the above (unoptimized) POC with varying interval durations and while transmitting 1 KB of data on our machine with a Core Ultra 155H processor. Figure 13 shows the results, achieving a maximum channel capacity of 16.62 bits per second. While this channel capacity is relatively low, it is in the same order of magnitude as that of prior covert channels that do not rely on monitoring shared microarchitectural structures or using timers [8], [9], [15].



(a) P-states (b) Idling hints

(c) P-states (fixed frequency) (d) Idling hints (fixed frequency)

Figure 14. Distribution of P-states and number of idling hints while processing 300 concurrent SIKE decapsulation requests for $m_i \neq m_{i-1}$ (low power) or $m_i = m_{i-1}$ (high power) cases. Figures 14a and 14b show the results under the default system configuration, and Figures 14c and 14d show the results under the setup with the reduced P-states ("fixed frequency"). Under both setups, the number of idling hints is power-dependent even though the P-states are power-independent.

6. Side-channel attacks

We now demonstrate that Thread Director enables a new class of remote power side-channel attacks where an adversary infers the power consumption of a victim solely by observing differences in the OS scheduling behavior. To this end, we present two end-to-end attacks. The first (Section 6.1) is a remote key extraction attack on a constant-time implementation of the SIKE cryptosystem. The second (Section 6.2) is a cross-origin pixel stealing attack leveraging SVG filters in Google Chrome. Unless otherwise stated, we run all attacks in the default system configuration, with both Hyper-threading and Turbo Boost enabled.

6.1. Remote timing attack on SIKE

Our first attack targets a constant-time implementation of the Supersingular Isogeny Key Encapsulation (SIKE) cryptosystem, which is a postquantum key encapsulation mechanism based on the Supersingular Isogeny Diffie-Hellman key exchange protocol. While SIKE was recently deprecated due to unrelated security concerns [65], [66], [67], we choose it as a target because it serves as a good benchmark to compare our attack against previous remote power side-channel attacks [9]. The attack exploits a power side-channel vulnerability that was discovered by Wang et. al [9] and, concurrently, De Feo et al. [68]. Through this vulnerability, an adversary with knowledge of the i least significant bits $(m_{i-1},...,m_0)$ of a secret key m can construct a challenge ciphertext such that SIKE's decapsulation function operates on a large number of zero values (resulting in a lower power consumption) if and only if $m_i \neq m_{i-1}$. We refer to the original papers [9], [68] for details on the cryptanalysis.

Threat model. As in the original attacks [9], [68], we assume a chosen-ciphertext attack model, where the adversary can send ciphertexts to the victim, which always tries to decapsulate them using its static secret key and replies with an acknowledgement indicating the establishment of a shared secret (but no other information). The attacker does not have any access to the victim's machine and can only remotely measure the time it takes to get replies from the victim. The goal of the attacker is to recover the secret key.

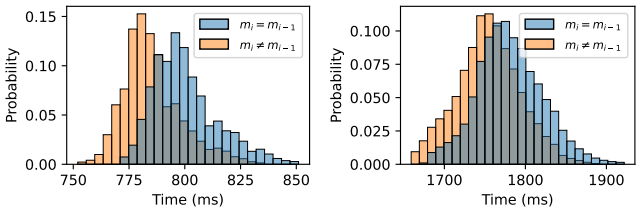
Experimental setup. We consider two system setups. In the first setup, we use the default system configuration. In the second setup, we limit the maximum P-states of P-cores and E-cores to 9 and 8, respectively, and reduce the power limit to 16 W (as opposed to the default of 28 W). We confirm that in this second setup, the P-states of all active P-cores and E-cores remain at 9 and 8, respectively, throughout the duration of the attack (as we show in Figure 14c)—preventing frequency side-channel leakage via Hertzbleed [9].

We target Cloudflare's constant-time implementation of SIKE [69] and the SIKE-751 parameter selection, which has 378-bit long secret keys. For the remote timing attack, we use the same setup as Hertzbleed. Specifically, we configure the server to accept decapsulation requests over HTTP and spawn a new thread to handle each request. The victim's and the attacker's machines are connected to the same network and the average round-trip time between them is 2.3 ms. The victim's machine uses our 155H processor.

Secret-dependent idling hints. We start by verifying that Thread Director gives fewer number of idling hints when the decapsulation function operates on a large number of zero values—resulting in lower power consumption—than when it does not. To this end, we use Section 3.2's methodology to sample the Thread Director table and the P-states of the active cores while processing 300 concurrent decapsulation requests to the SIKE server. We run this experiment 1,000 times using challenge ciphertexts that induce zero values, and another 1,000 times using ciphertexts that do not. Since we run the experiment with Hyper-threading enabled, we report idling hints at the granularity of each hyperthread.

Figures 14a and 14b show the results under the default system configuration. The results show that the mean P-state is the same but the average number of idling hints is lower when the decapsulation function operates on many zero values ($m_i \neq m_{i-1}$) than when it does not ($m_i = m_{i-1}$). As a result, the P-core usage of the SIKE is also lower under these conditions. We observe a similar power-dependent number of idling hints when running the experiments in the setup with the reduced P-states (Figure 14d) even though the P-states of the active cores is constant (Figure 14c). Neither setup shows frequency side-channel leakage.

Remote key recovery. We now demonstrate that the secret-dependent number of idling hints observed above translates to a timing difference that is observable remotely *despite the absence of frequency side-channel leakage*.



(a) Response time (b) Response time (limited freq.)

Figure 15. Distribution of the timings measured by the attacker during the remote key extraction attack, grouped based on whether they triggered a large number of zero values ($m_i \neq m_{i-1}$) or not ($m_i = m_{i-1}$). Each sample represents the time required to complete 300 concurrent SIKE decapsulation requests (all using the same challenge ciphertext).

As in the original attack [9], we set up the SIKE server with a randomly generated key m, and we configure the attacker to recover the first 364 bits of this key via an adaptive chosen ciphertext attack and the last 14 bits by brute-force search. Having recovered the first i bits of m, the attacker sends 300 concurrent decapsulation requests to the server with a challenge ciphertext that should trigger a large number of zero values if and only if $m_i \neq m_{i-1}$. The attacker measures the time to receive responses for all 300 requests and repeats this measurement 70 times. If the average of the observed times is shorter than a baseline (experimentally established through profiling), the adversary concludes that $m_i \neq m_{i-1}$; otherwise, they conclude that $m_i = m_{i-1}$. The attacker then proceeds to the next bit. If 20 consecutive bit positions show no timing reduction, we assume a bit was recovered incorrectly and backtrack.

Figures 15a (default system configuration) and 15b (reduced P-states configuration) show the distribution of the 300-connection timing samples collected during the attack, grouped according to whether the challenge ciphertext triggered a large number of zero values ($m_i \neq m_{i-1}$) or not ($m_i = m_{i-1}$). These results demonstrate that secret-dependent differences in the number of active cores are remotely observable even in the absence of frequency side-channel leakage. In the default system configuration, the attack successfully recovered the full secret key in 7 h and 8 min after needing to backtrack twice. In the configuration with reduced P-states, the attack successfully recovered the full secret key in 38 h and 21 min after needing to backtrack 5 times. In contrast, the remote key extraction attack demonstrated in the original Hertzbleed paper required 36 h to leak the full cryptographic key (on an older x86 processor under the default system configuration) [9].

6.2. Pixel stealing attack

Our second POC demonstrates a cross-origin pixel stealing attack on Google Chrome version 135 (the latest at the time of writing). The attack is similar to the cross-component Hertzbleed attack demonstrated by Wang et al. [11] but is the first to leak due to OS scheduling behavior.

(a) original (b) recovered

Figure 16. Result of our pixel stealing POC on Google Chrome, which retrieves the checkerboard residing in a cross-origin iframe.

Attack setup. Our POC design is similar to the one used in prior work [11], [12], [37], [38], [39], [40], [49]. To steal a pixel, we first scale a cross-origin 1x1 target pixel into a 3000x3000 `iframe`. We also apply `feColorMatrix` and `feComponentTransfer` in this process to binarize a pixel of any color to black or white. We use `feComposite` to compose a random image and the target iframe. We then apply a chain of `feGaussianBlur` filters which applies a two-dimensional Gaussian function on this composed iframe. To continuously repeat this application of SVG filters, we loop through `requestAnimationFrame` calls. We make the GPU idle for 2.1 s between each pixel for the processor to stop throttling. Based on whether the 1x1 target pixel was black or white, applying the above stack of SVG filters results in a large difference in GPU power consumption, changing when the processor throttles again.

Inferring the GPU power consumption via core usage. Shortly after activating the GPU with the above filter stack, the processor begins throttling. After this point, Thread Director gives idling hints on all cores except for the last one or two E-cores. As we discuss in Sections 3.2 and 4.2, the time when these idling hints appear depends on the GPU power consumption. In our framework specifically, we observe that only white pixels cause the processor to throttle (and thus exhibit idling hints) within 3.3 s of launching the filter stack. Thus, an attacker can infer the target pixel color by checking for the presence of idling hints in the 3.1-3.3 s window after the SVG filter stack begins. This can be observed even from Javascript, since the idling hints cause a large reduction in the performance of CPU workloads. Specifically, we infer the presence of idling hints by spawning 20 Javascript threads, matching the number of hyperthreads on our processor, and measuring the cumulative number of increments they can do in the above 200 ms window. Using this setup, the number of increments is 2.4× larger when the target pixel is black pixels (no idling hints) than when it is white (idling hints).

To evaluate our attack, we embed a checkerboard in a cross-origin iframe and demonstrate that an attacker can recover it in Figure 16. Our attack achieves a 96.3% accuracy and takes 7.1 s to leak each pixel, which is faster than the one reported by Taneja et al. [12] on Intel GPUs (22.6 s per pixel) and slightly slower than the one reported by Wang et al. [11] on Intel GPUs (1-3 s per pixel). We expect that this time could be reduced with careful optimization.

7. Discussion and future work

Other system-on-a-chip components. While this work only considers CPU and GPU workloads, an outstanding question is whether our attack can be generalized to leak secrets from other system on a chip (SoC) components. For example, Intel mentions that Thread Director hints may also depend on the power consumption of the Neural Processing Unit (NPU), which is used by AI workloads [70]. Investigating the feasibility of our attack on the NPU (and potentially other on-chip accelerators) requires future work. However, the techniques we use to build our cross-component attacks should be applicable to components beyond the GPU.

Other microarchitectures and OSs. Intel claims that, while the way Thread Director exposes information to the OS has remained consistent across all Intel processors since Alder Lake, the internal power management algorithms determining what information Thread Director provides have changed with Meteor Lake [54], [70]. Indeed, we were unable to replicate our attacks and trigger idling hints on an Intel Core i9-13900 Raptor Lake processor. We hypothesize it should be possible to port our attacks to Intel microarchitectures released after Meteor Lake (e.g., Lunar Lake and Arrow Lake), but doing so may require additional tuning.

Two open questions are whether our attacks can be generalized to other OSs and to non-Intel processors. We believe that the answer is primarily "yes". First, Intel is working to add Thread Director support to other OSs, such as Linux [71]. Second, other vendors are starting to offer architectures with Thread Director-like optimizations. For example, AMD recently submitted for review a series of patches implementing an AMD Hardware Feedback Interface driver designed to provide "behavioral classification and a dynamically updated ranking table for the scheduler to use when choosing cores for tasks" [72]. Similarly, comments in the open-sourced macOS kernel code suggest that the scheduler may use "signals like thermal levels for optimal power/perf tradeoffs for a platform" [73] and that some cores may be "derecommended due to thermals" [74]. We leave a detailed investigation of these optimizations to future work.

Mitigations. One possible mitigation to our attacks is to disable Thread Director via the `IA32_HW_FEEDBACK_CONFIG` and `IA32_HW_FEEDBACK_THREAD_CONFIG` MSRs [21]. We verified that writing 0 to these MSRs effectively prevents Thread Director table and class ID updates, respectively. However, this approach requires privileged access, applies system-wide, may result in suboptimal performance (according to Intel—some workloads enjoy 14% performance improvement with Thread Director enabled [61]), and leaves the system still vulnerable to Hertzbleed.

Since both our attack and Hertzbleed rely on the processor throttling, a more general system-level mitigation against both attacks is to reduce the likelihood that the processor throttles. One way to achieve this is to lower the maximum allowed CPU P-state, as recommended by prior work and the cryptographic community [9], [12], [75], [76]. However, this approach incurs high performance overheads and may not prevent attacks when the CPU power budget is severely constrained (e.g., due to high-power GPU workloads or custom power limits, as we show in Section 6).

In use cases where doing so is possible, performing frequent key refreshes (as recommended by prior work [10]) would also mitigate our attack, since remote power side-channel attacks require the ability to repeatedly initiate operations with the same secret key.

In the long term, research is needed to develop general, practical mechanisms to mitigate remote power side-channel attacks purely from software. Physical power side-channel defenses such as masking [77], [78] would prevent our attacks, but masking is program-specific and incurs significant performance overheads. As prior work notes, a more practical approach could involve taking advantage of the lower resolution of remote power side-channel attacks compared to physical ones [9], [11]. For example, ensuring that average power usage in a given time period (e.g., 1 ms) is independent of secret data might be a sufficient mitigation. However, how to apply this approach to arbitrary programs remains an open research question.

Finally, our pixel stealing attack can be mitigated by restricting sensitive data from being displayed inside cross-origin iframes (cf. Section 2.4).

8. Related work

Hertzbleed [9] was the first work to demonstrate that—when the processor hits certain thermal or power limits and starts throttling—the processor's CPU frequency becomes dependent on the CPU (and GPU [11]) power consumption. Our attack similarly relies on the processor throttling but exploits differences in the CPU scheduling behavior rather than frequency scaling. Specifically, we show that when the active cores are unable to run above certain frequencies, Meteor Lake's power management algorithm prefers reducing the number of active cores rather than further lowering the CPU frequency. This can lead to power-dependent scheduling leakage even in the absence of frequency leakage.

Our work is also related to the work from Taneja et al., who demonstrated that when the processor throttles, the GPU frequency depends on the GPU power consumption, enabling pixel stealing attacks on Intel and other GPUs [12]. Our pixel stealing attacks also rely on the GPU downclocking its frequency, but do not require the ability to measure the GPU rendering time. Additionally, our pixel stealing attack is faster on Intel GPUs, where Taneja et al.'s attack leaked 1 pixel every 22.6 s (with 77% accuracy).

Finally, our work is related to the rich literature on pixel stealing attacks, which we covered in Section 2.4, and to recent research on remote power side-channel attacks, which we covered in Section 1. Concurrent work from Oberhuber et al. demonstrates that both these attacks can be performed by exploiting unprivileged APIs provided by the Android sensor framework [79]. Our work is the first to demonstrate that both these attacks can be performed just by monitoring differences in the OS scheduling behavior.

9. Conclusion

We demonstrated that the power management algorithms of modern Intel architectures enable a new class of remote power side-channel attacks that work even in the absence of frequency side-channel leakage. The root cause of these attacks is hardware-guided scheduling due to Intel Thread Director. We showed that this optimization depends on the processor's power consumption and leads to behaviors—such as variations in the number of active cores—that can be observed via remote-timing analysis. Our attacks are up to 5× faster than those previously demonstrated on older x86 processors and remain effective even when the frequency of the active CPU cores is constant. Thus, future mitigations should not just focus on frequency leakage and must consider all sources of remote power side-channel leakage.

Acknowledgment

We thank the reviewers for their feedback and Intel PSIRT for their collaboration during the disclosure process. This work was supported in part by a CyLab Seed grant and a Korea Foundation for Advanced Studies fellowship.

References

[1] S. Mangard, E. Oswald, and T. Popp, *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. Springer Science & Business Media, 2008, vol. 31.

[2] S. Mangard, "A simple power-analysis (SPA) attack on implementations of the AES key expansion," in *ICISC*, 2002.

[3] T. Messerges, "Using second-order power analysis to attack DPA resistant software," in *CHES*, 2000.

[4] T. Messerges, E. Dabbish, and R. Sloan, "Investigations of power analysis attacks on smartcards," in *USENIX Smartcard*, 1999.

[5] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *CRYPTO*, 1999.

[6] M. Zhao and E. Suh, "FPGA-based remote power side-channel attacks," in *S&P*, 2018.

[7] H. Mantel, J. Schickel, A. Weber, and F. Weber, "How secure is green IT? the case of software-based energy side channels," in *ESORICS*, 2018.

[8] M. Lipp, A. Kogler, D. Oswald, M. Schwarz, C. Easdon, C. Canella, and D. Gruss, "PLATYPUS: Software-based power side-channel attacks on x86," in *S&P*, 2021.

[9] Y. Wang, R. Paccagnella, E. He, H. Shacham, C. W. Fletcher, and D. Kohlbrenner, "Hertzbleed: Turning power side-channel attacks into timing attacks on x86," in *USENIX Security*, 2022.

[10] C. Liu, A. Chakraborty, N. Chawla, and N. Roggel, "Frequency throttling side-channel attack," in *CCS*, 2022.

[11] Y. Wang, R. Paccagnella, A. Wandke, Z. Gang, G. Garrett-Grossman, C. W. Fletcher, D. Kohlbrenner, and H. Shacham, "DVFS frequently leaks secrets: Hertzbleed attacks beyond SIKE, cryptography, and CPU-only data," in *S&P*, 2023.

[12] H. Taneja, J. Kim, J. J. Xu, S. van Schaik, D. Genkin, and Y. Yarom, "Hot pixels: Frequency, power, and temperature attacks on GPUs and ARM SoCs," in *USENIX Security*, 2023.

[13] Z. Zhang, S. Liang, F. Yao, and X. Gao, "Red alert for power leakage: Exploiting Intel RAPL-induced side channels," in *ASIACCS*, 2021.

[14] Y. Cohen, K. S. Tharayil, A. Haenel, D. Genkin, A. D. Keromytis, Y. Oren, and Y. Yarom, "HammerScope: Observing DRAM power consumption using Rowhammer," in *CCS*, 2022.

[15] M. Lipp, D. Gruss, and M. Schwarz, "AMD prefetch attacks through power and time," in *USENIX Security*, 2022.

[16] A. Kogler, J. Juffinger, L. Giner, L. Gerlach, M. Schwarzl, M. Schwarz, D. Gruss, and S. Mangard, "Collide+Power: Leaking inaccessible data with software-based power side channels," in *USENIX Security*, 2023.

[17] M. Botvinnik, T. Laor, T. Rokicki, C. Maurice, and Y. Oren, "The finger in the power: How to fingerprint PCs by monitoring their power consumption," in *DIMVA*, 2023.

[18] Intel, "Running average power limit energy reporting / cve-2020-8694 , cve-2020-8695 / intel-sa-00389," https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/advisory-guidance/running-average-power-limit-energy-reporting.html, accessed on Oct 28, 2024.

[19] L. Brown, "powercap: restrict energy meter to root access," https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=949dd0104c496fa7c14991a23c03c62e44637e71, 2020, accessed on Oct 28, 2024.

[20] E. Rotem, A. Yoaz, L. Rappoport, S. J. Robinson, J. Y. Mandelblat, A. Gihon, E. Weissmann, R. Chabukswar, V. Basin, R. Fenger *et al.*, "Intel Alder Lake CPU architectures," *IEEE Micro*, vol. 42, no. 3, 2022.

[21] Intel, *Intel 64 and IA-32 Architectures Software Developer's Manual*, October 2024.

[22] Microsoft, "Scheduling priorities - Win32 apps — Microsoft Learn," https://learn.microsoft.com/en-us/windows/win32/procthread/scheduling-priorities, accessed on Oct 28, 2024.

[23] "SVG and CSS filters can leak cross-origin data via iframes," https://issues.chromium.org/issues/401081629, 2025, accessed on Apr 1 2025.

[24] H. Chung, M. Kang, and H.-D. Cho, "Heterogeneous multi-processing solution of Exynos 5 Octa with ARM big.LITTLE technology," Samsung, White Paper, 2012.

[25] Apple, "Apple unleashes M1," https://www.apple.com/newsroom/2020/11/apple-unleashes-m1/, 2020, accessed on May 2 2024.

[26] Intel, "Lakefield: Hybrid CPU with Foveros technology," https://www.intel.com/content/www/us/en/newsroom/resources/lakefield.html, 2020, accessed on May 2 2024.

[27] ——, "Intel unveils 12th Gen Intel Core, launches world's best gaming processor, i9-12900K," https://www.intel.com/content/www/us/en/newsroom/news/12th-gen-core-processors.html, 2021, accessed on May 2 2024.

[28] AMD, "AMD unveils Ryzen 8000G series processors: Zen 4 APUs for desktop with Ryzen AI," https://www.anandtech.com/show/21208/amd-unveils-ryzen-8000g-series-processors-zen-4-apus-for-desktop-with-ryzen-ai, 2024, accessed on May 2 2024.

[29] Intel, "Meteor Lake architecture overview," https://www.intel.com/content/www/us/en/content-details/788851/meteor-lake-architecture-overview.html, 2023, accessed on May 2 2024.

[30] ——, *Intel 64 and IA-32 Architectures Optimization Reference Manual*, April 2024.

[31] N. Rukmabhatla, R. Chabukswar, S. Gohad, and M. Chynoweth, "Intel performance hybrid architecture & software optimizations," Intel, Tech. Rep., 2021.

[32] Intel, "What is clock speed?" https://www.intel.com/content/www/us/en/gaming/resources/cpu-clock-speed.html, accessed on Nov 7, 2024.

[33] R. J. Wysocki, "`intel_pstate` CPU performance scaling driver," https://docs.kernel.org/admin-guide/pm/intel_pstate.html, accessed on Nov 7, 2024.

[34] Intel, "Frequency throttling side channel software guidance for cryptography implementations," https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/best-practices/frequency-throttling-side-channel-guidance.html, 2022, accessed on Nov 7, 2024.

[35] Linux, "`rapl.c`," https://github.com/torvalds/linux/blob/master/arch/x86/events/rapl.c, accessed on Oct 29, 2024.

[36] "W3C scalable vector graphics (SVG) 1.1 (second edition)," https://www.w3.org/TR/SVG11/, 2011, accessed on May 2 2024.

[37] P. Stone, "Pixel perfect timing attacks with HTML5," Context Information Security, White Paper, 2013.

[38] R. Kotcher, Y. Pei, P. Jumde, and C. Jackson, "Cross-origin pixel stealing: Timing attacks using CSS filters," in *CCS*, 2013.

[39] M. Andrysco, D. Kohlbrenner, K. Mowery, R. Jhala, S. Lerner, and H. Shacham, "On subnormal floating point and abnormal timing," in *S&P*, 2015.

[40] D. Kohlbrenner and H. Shacham, "On the effectiveness of mitigations against floating-point timing channels," in *USENIX Security*, 2017.

[41] "Canvas composite operations and CSS blend modes leak cross-origin data via timing attacks," https://issues.chromium.org/issues/40086984, 2017, accessed on May 2 2024.

[42] "Cross-origin pixel reading and history sniffing via SVG filter timing attack," https://issues.chromium.org/issues/40086661, 2017, accessed on May 2 2024.

[43] "SVG filter timing attack," https://bugzilla.mozilla.org/show_bug.cgi?id=711043, 2011, accessed on May 2 2024.

[44] "W3C filter effects module level 1," https://www.w3.org/TR/filter-effects-1/#priv-sec, 2018, accessed on May 2 2024.

[45] Mozilla, "Firefox rolls out Total Cookie Protection by default to more users worldwide," https://blog.mozilla.org/en/products/firefox/firefox-rolls-out-total-cookie-protection-by-default-to-all-users-worldwide/, 2022, accessed on Apr 1 2025.

[46] J. Hofmann and T. Huang, "Introducing state partitioning," https://hacks.mozilla.org/2021/02/introducing-state-partitioning/, 2021, accessed on Apr 1 2025.

[47] "Feature: Cookies default to SameSite=Lax," https://chromestatus.com/feature/5088147346030592, 2020, accessed on Apr 1 2025.

[48] "[meta] enable sameSite=lax by default," https://bugzilla.mozilla.org/show_bug.cgi?id=1617609, 2020, accessed on Apr 1 2025.

[49] Y. Wang, R. Paccagnella, Z. Gang, W. R. Vasquez, D. Kohlbrenner, H. Shacham, and C. W. Fletcher, "GPU.zip: On the side-channel implications of hardware-based graphical data compression," in *S&P*, 2024.

[50] S. O'Connell, L. A. Sour, R. Magen, D. Genkin, Y. Oren, H. Shacham, and Y. Yarom, "Pixel thief: Exploiting SVG filter leakage in Firefox and Chrome," 2024.

[51] PhyxionNL, "LibreHardwareMonitor," https://github.com/LibreHardwareMonitor/LibreHardwareMonitor, 2024, accessed on Oct 29, 2024.

[52] I. Cutress, "Intel 12th gen Core Alder Lake for desktops: Top SKUs only, coming November 4th," https://www.anandtech.com/show/16959/intel-innovation-alder-lake-november-4th/3, 2021, accessed on Nov 7, 2024.

[53] Intel, "Intel Core Ultra processors," https://download.intel.com/newsroom/2023/ai/ai-everywhere-2023/Intel-Core-Ultra-Processors-Media-Presentation.pdf, 2023, accessed on Nov 7, 2024.

[54] L. Waldock, "How does Thread Director manage Intel Meteor Lake?" https://www.kitguru.net/components/cpu/leo-waldock/how-does-thread-director-manage-intel-meteor-lake/, accessed on Nov 7, 2024.

[55] M. Gupta, "Optimizing software on Lunar Lake processor hybrid architecture — tech talk — innovation selects," https://www.youtube.com/watch?v=1ULS_LLHTnc, accessed on Nov 7, 2024.

[56] D. R. Subbareddy, G. N. Srinivasa, E. Gorbatov, S. D. Hahn, D. A. Koufaty, P. Brett, and A. Prabhakaran, "Apparatus and method for intelligently powering heterogeneous processor components," Patent US9672046B2, 2012.

[57] S. Jahagirdar, A. Koker, Y. Harel, K. Brand, C. Gurram, E. Finley, B. Borole, and C. N. Rodriguez, "Resource load balancing based on usage and power limits," Patent US10983581B2, 2017.

[58] "EEVDF scheduler," https://docs.kernel.org/scheduler/sched-eevdf.html, accessed on Oct 28, 2024.

[59] "CFS scheduler," https://docs.kernel.org/scheduler/sched-design-CFS.html, accessed on Oct 28, 2024.

[60] F. Zhou, M. Goel, P. Desnoyers, and R. Sundaram, "Scheduler vulnerabilities and attacks in cloud computing," *Journal of Computer Security*, vol. 21, no. 4, 2013.

[61] Z. Liu, "[RFC 00/26] Intel Thread Director virtualization," https://lwn.net/Articles/960772/, 2024, accessed on Nov 7, 2024.

[62] D. Gullasch, E. Bangerter, and S. Krenn, "Cache games - bringing access-based cache attacks on AES to practice," in *S&P*, 2011.

[63] T. Allan, B. B. Brumley, K. Falkner, J. Van de Pol, and Y. Yarom, "Amplifying side channels through performance degradation," in *CCS*, 2016.

[64] Y. Zhu, B. Chen, Z. N. Zhao, and C. W. Fletcher, "Controlled preemption: Amplifying side-channel attacks from userspace," in *ASPLOS*, 2025.

[65] W. Castryck and T. Decru, "An efficient key recovery attack on SIDH," in *EUROCRYPT*, 2023.

[66] D. Robert, "Breaking SIDH in polynomial time," in *EUROCRYPT*, 2023.

[67] L. Maino, C. Martindale, L. Panny, G. Pope, and B. Wesolowski, "A direct key recovery on SIDH," in *EUROCRYPT*, 2023.

[68] L. De Feo, N. El Mrabet, A. Genêt, N. Kaluđerović, N. L. de Guertechin, S. Pontié, and É. Tasso, "SIKE channels: Zero-value side-channel attacks on SIKE," *TCHES*, vol. 2022, 2022.

[69] A. Faz-Hernández and K. Kwiatkowski, *Introducing CIRCL: An Advanced Cryptographic Library*, Cloudflare, 2019, https://github.com/cloudflare/circl. Accessed on Nov 7, 2024.

[70] R. Chabukswar, "Intel Thread Director: Lunar Lake optimizations explained — talking tech — Intel technology," https://www.youtube.com/watch?v=agJwHsShFd8, accessed on Nov 7, 2024.

[71] R. Neri, "[PATCH v4 00/24] sched: Introduce classes of tasks for load balance," https://lkml.org/lkml/2023/6/13/8, accessed on Oct 28, 2024.

[72] M. Larabel, "AMD hardware feedback interface "HFI" driver updated for heterogeneous CPUs," https://www.phoronix.com/news/AMD-HFI-Linux-Driver-v2, 2024, accessed on Nov 7, 2024.

[73] Apple, "xnu/osfmk/kern/sched_prim.c," https://github.com/apple-oss-distributions/xnu/blob/xnu-11215.1.10/osfmk/kern/sched_prim.c, 2024, accessed on Nov 7, 2024.

[74] ——, "xnu/tests/sched/all_cores_running.c," https://github.com/apple-oss-distributions/xnu/blob/xnu-11215.1.10/tests/sched/all_cores_running.c, 2024, accessed on Nov 7, 2024.

[75] PQC Email List, "HertzBleed : power side channel attacks on SIKE," https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/jAj_8Hreqc0, 2022, accessed on Oct 28, 2024.

[76] D. J. Bernstein, "Timing attacks: Overclocking FAQ," https://timing.attacks.cr.yp.to/overclocking.html, 2022, accessed on Oct 28, 2024.

[77] S. Chari, C. S. Jutla, J. R. Rao, and P. Rohatgi, "Towards sound approaches to counteract power-analysis attacks," in *CRYPTO*, 1999.

[78] L. Goubin and J. Patarin, "DES and differential power analysis the "duplication" method," in *CHES*, 1999.

[79] M. Oberhuber, M. Unterguggenberger, L. Maar, A. Kogler, and S. Mangard, "Power-related side-channel attacks using the Android sensor framework," in *NDSS*, 2025.