Agile C-states: A Core C-state Architecture for Latency Critical Applications Optimizing both Transition and Cold-Start Latency

GEORGIA ANTONIOU, Computer Science, University of Cyprus, Nicosia, Cyprus
DAVIDE BARTOLINI, Computing Systems Lab, Zurich Research Center, Huawei Technologies Switzerland AG, Zurich, Switzerland
HARIS VOLOS, Computer Science, University of Cyprus, Nicosia, Cyprus
MARIOS KLEANTHOUS, Computer Science, University of Cyprus, Nicosia, Cyprus
ZHE WANG, Computing Systems Lab, Zurich Research Center, Huawei Technologies Switzerland AG, Zurich, Switzerland
KLEOVOULOS KALAITZIDIS, Computing Systems Lab, Zurich Research Center, Huawei Technologies Switzerland AG, Zurich, Switzerland
TOM ROLLET, Computing Systems Lab, Zurich Research Center, Huawei Technologies Switzerland AG, Zurich, Switzerland
ZIWEI LI, Huawei Technologies Co Ltd, Shenzhen, China
ONUR MUTLU, Information Technology and Electrical Engineering, ETH Zurich, Zurich, Switzerland
YIANNAKIS SAZEIDES, Computer Science, University of Cyprus, Nicosia, Cyprus
JAWAD HAJ YAHYA, Rivos Inc, Mountain View, California, USA

Latency-critical applications running in modern datacenters exhibit irregular request arrival patterns and are implemented using multiple services with strict latency requirements (30–250μs). These characteristics render existing energy-saving idle CPU sleep states ineffective due to the performance overhead caused by the state's transition latency. Besides the state transition latency, another important contributor to the performance overhead of sleep states is the cold-start latency, or in other words, the time required to

J. Haj Yahya is currently affiliated with Rivos Inc.

This project received funding from a Huawei grant and the European Union's Horizon 2020 research and innovation programme under Marie Skłodowska-Curie grant agreement 101029391.

Authors' Contact Information: Georgia Antoniou, Computer Science, University of Cyprus, Nicosia, Cyprus; e-mail: ganton12@cs.ucy.ac.cy; Davide Bartolini, Computing Systems Lab, Zurich Research Center, Huawei Technologies Switzerland AG, Zurich, Switzerland; e-mail: davide.basilio.bartolini@huawei.com; Haris Volos, Computer Science, University of Cyprus, Nicosia, Cyprus; e-mail: hvolos01@ucy.ac.cy; Marios Kleanthous, Computer Science, University of Cyprus, Nicosia, Cyprus; e-mail: marios@kleanthous.info; Zhe Wang, Computing Systems Lab, Zurich Research Center, Huawei Technologies Switzerland AG, Zurich, Switzerland; e-mail: wangzhe59@huawei.com; Kleovoulos Kalaitzidis, Computing Systems Lab, Zurich Research Center, Huawei Technologies Switzerland AG, Zurich, Switzerland; e-mail: kleovoulos.kalaitzidis@huawei.com; Tom Rollet, Computing Systems Lab, Zurich Research Center, Huawei Technologies Switzerland AG, Zurich, Switzerland; e-mail: tom.rollet@huawei.com; Ziwei Li, Huawei Technologies Co. Ltd., Shenzhen, Guangdong, China; e-mail: liziwei13@huawei.com; Onur Mutlu, Information Technology and Electrical Engineering, ETH Zurich, Zurich, Switzerland; e-mail: omutlu@gmail.com; Yiannakis Sazeides, Computer Science, University of Cyprus, Nicosia, Cyprus; e-mail: yanos@cs.ucy.ac.cy; Jawad Haj Yahya, Rivos Inc, Mountain View, California, USA; e-mail: jawadhy@gmail.com.

ACM 1544-3566/2024/11-ART66
https://doi.org/10.1145/3674734

warm up the microarchitectural state (e.g., cache contents, branch predictor metadata) that is flushed or discarded when transitioning to a lower-power state. Both the transition latency and cold-start latency can be particularly detrimental to the performance of latency critical applications with short execution times. While prior work focuses on mitigating the effects of transition and cold-start latency by optimizing request scheduling, in this work we propose a redesign of the core C-state architecture for latency-critical applications. In particular, we introduce C6Awarm, a new Agile core C-state that drastically reduces the performance overhead caused by idle sleep state transition latency and cold-start latency while maintaining significant energy savings. C6Awarm achieves its goals by (1) implementing medium-grained power gating, (2) preserving the microarchitectural state of the core, and (3) keeping the clock generator and PLL active and locked. Our analysis for a set of microservices based on an Intel Skylake server shows that C6Awarm manages to reduce the energy consumption by up to 70% with limited performance degradation (at most 2%).

CCS Concepts: • **Hardware → Enterprise level and data centers power issues**; *Chip-level power issues;* • **Software and its engineering** → *Power management;* • **Computer systems organization** → *Architectures*;

Additional Key Words and Phrases: Power management, idle states, C-states, datacenters, microservices

ACM Reference Format:
Georgia Antoniou, Davide Bartolini, Haris Volos, Marios Kleanthous, Zhe Wang, Kleovoulos Kalaitzidis, Tom Rollet, Ziwei Li, Onur Mutlu, Yiannakis Sazeides, and Jawad Haj Yahya. 2024. Agile C-States: A Core C-State Architecture for Latency Critical Applications Optimizing Both Transition and Cold-Start Latency. *ACM Trans. Arch. Code Optim.* 21, 4, Article 66 (November 2024), 26 pages. https://doi.org/10.1145/3674734

1 Introduction

Servers in today's datacenters are utilized inefficiently when running latency-critical applications based on microservices. The load unpredictability [11, 51, 64] characterizing latency-critical applications makes it unfeasible for the system to use energy-saving techniques such as CPU idle sleep states [11, 64]. Even if the application load could be predicted and energy-saving techniques could be utilized, the substantial performance overhead introduced by CPU idle sleep states (Table 1) poses a serious threat to the performance of latency-critical applications (30–250μs [10, 72, 97]). Consequently, datacenter operators opt to deactivate energy-saving features such as idle CPU sleep states, thereby removing any potential performance overheads [56, 60] at the cost of higher power. The gravity of this issue becomes more pronounced when considering that servers running latency-critical applications frequently operate at low utilization levels to keep tail latency under control [7, 64, 65]. As a result, even though a system is mostly idle, it consumes energy at a rate similar to when it is fully active.

Core C-states [26] are a popular power management method whereby an idle core enters a power state to reduce or even eliminate its power consumption. Table 1 presents the core C-states of an Intel Skylake-based server processor, along with their power consumption and transition latency. Transition latency represents the time the core requires to transition from one state to another. During the transition, the core is unable to perform any work. For example, it is estimated that $C6$ requires 133μs transition time. As a result, entry/exit latencies can degrade the performance of services that have microsecond latency requirements, such as user-facing applications [51]. Additionally, for a transition to result in power savings, the core needs to reside in the idle state for a minimum amount of time, referred to as *target residency time*. Target residency time represents the minimum time required by the system to amortize the transition energy overhead.

The bursty dynamic behavior [11, 51, 64] of latency-critical applications based on microservices makes the idle time unpredictable and prevents a core from entering deep sleep states. Even if the

Table 1. C-States on Intel Skylake Server Cores [45] and Our New *C6Awarm* and *C6AwarmE* C-States

C-State	Transition Time	Residency Time	Power per Core
C0 (*P1*)	N/A	N/A	~4W
C0 (*Pn*)	N/A	N/A	~1W
C1 (*P1*)	2µs	2µs	1.44W
C6Awarm (*P1*)	2.09µs	2µs	~0.35W
C1E (*Pn*)	10µs	20µs	0.88W
C6AwarmE (*Pn*)	10.09µs	20µs	~0.27W
C6	133µs	600µs	~0.1W

power management system could predict the idle time, deep core sleep states would not be utilized because their transition latency is close to the QoS constraints for latency-critical applications. For example, Memcached is a microservice with typical QoS requirements of 30–250μs. An entry/exit from a deep sleep state (i.e., C6, 133μs) consumes at least 50% of the latency budget, increasing the risk for QoS violations. Moreover, even if QoS violations could be avoided, the target residency for deep sleep states is often longer than the typical core idle period for Memcached [3].

Besides unpredictability of idleness and transition latency, the time required to *warm up* the core resources after power gating them further increases the performance overhead and reduces the opportunity to benefit from entering deep core sleep states. We refer to the time needed to warm up the data arrays of the core as cold-start latency [2, 4, 10, 12].

The operating system (e.g., Linux) decides which C-state to enter based on an *idle governor*. Current idle governors only consider C-state transition latency and target residency and disregard cold-start latency. This is because idle governors were designed for client devices and applications [91, 92] and assume that workloads typically exhibit relatively long and predictable active and idle periods. This behavior is not representative for microservices because (1) the computation time per microservice invocation often falls in the 30–250μs range, and (2) servers that run microservices typically operate at low utilization [7, 64, 65] to harness tail latency. These characteristics underline the significance of the cold-start latency due to deep core idle states.

Prior work proposes solutions to enable the effective use of existing power management techniques like C-states and **Dynamic Voltage Frequency Scaling (DVFS)**. The solutions include fine-grained latency-aware DVFS or/and core C-states management and core consolidation [11, 54, 63, 64, 97, 98]. Another set of works [2, 4, 10, 12, 83] tries to quantify the cold-start latency either through power gating [2, 4] or process migration [12, 83] with the aim to model [4, 10] it or eliminate it through prefetching [83] or improve idle governor decisions [2, 4]. Finally, another group of works focuses on improving idle time predictability [15, 86] through the use of machine learning. Our work differs from the previous ones because (1) it optimizes power consumption, (2) the analysis focuses on both transition and cold-start latency of core C-states, and (3) the proposed solution is a hardware-based solution incorporated into the current core C-state architecture.

We propose C6Awarm, a new deep core C-state for processors in datacenters running latency-critical applications. C6Awarm goal is to reduce both transition and cold-start latency while achieving significant energy savings. C6Awarm achieves this goal by using the following techniques. First, instead of switching off the entire core in deep sleep states, C6Awarm uses medium-grained power gates to maintain the core context (e.g., Control Registers, Patch RAM, microcode firmware) in place, avoiding the time to save/restore from an external SRAM. Second, instead of shutting down the structures of the core containing a program's architectural and microarchitectural state (e.g., branch predictors, TLBs, prefetchers, L1D, L1I, L2), C6Awarm keeps them power

Table 2. Microarchitectural State for Skylake Server Core C-States [44] Including AW's New *C6Awarm* and *C6AwarmE* C-States

C-State	Clocks	ADPLL	L1/L2 Cache	Voltage	Context
C0	Running	On	Coherent	Active	Maintained
C1	Stopped	On	Coherent	Active	Maintained
C6Awarm	Stopped	On	Coherent	PG/Ret/Active	In-place S/R
C1E	Stopped	On	Coherent	Min V/F	Maintained
C6AwarmE	Stopped	On	Coherent	PG/Ret/Min V/F	In-place S/R
C6	Stopped	Off	Flushed	Shut-off	S/R SRAM

ungated, eliminating both the time required to power gate them and the time required to warm them up. Additionally, to eliminate the leakage power of the power ungated structures, C6Awarm implements a sleep mode in caches and predictors. Third, C6Awarm clock gates the clock distribution and keeps the ADPLL on and locked.

This article extends our previous work [92] with the following contributions:

(1) Analysis of core C-state transition and cold-start latency on microservices. The analysis reveals that the data caches and frontend resources contribute the most to the transition and cold-start latency, respectively.
(2) Redesign of core C-state architecture. C6Awarm is the first core C-state that optimizes both transition and cold-start latency, directly targeting datacenter servers running latency-critical applications.
(3) Analytical quantification of the performance, power savings, and area overhead of C6Awarm using publicly available models and data.
(4) Experimental evaluation of C6Awarm on both two- and three-tier microservice-based benchmarks. Based on the evaluation, C6Awarm can reduce energy consumption by up to 70% with minimal performance degradation (up to 2%).

2 Sources of Performance Degradation for Deep Sleep States

Before examining the different types of latencies that core C-states cause, we first present their architecture and entry/exit flow. Core C-states are power-saving states that enable the core to reduce its power consumption during idle periods. Modern processors support various C-states—for example, Intel's Skylake architecture offers the following four: $C0$, $C1$, $C1E$, and $C6$ [26, 32]. Core C-state $C0$ represents the active state and $C6$ the deepest sleep state during which the core is power gated. $C1$ and $C1E$ are intermediate sleep states during which the core is either clock gated or the frequency of the core is set to minimum. More details regarding the microarchitectural state of the core at each C-state can be found in Table 2 (along with the proposed C-states *C6Awarm*, *C6AwarmE*). The C1/C1E and C6 entry and exit flows are shown in Figure 1(a) and (b), respectively, and are discussed in detail in other works [26, 32, 79, 84, 85].

2.1 Transition Latency

Transition latency is the time required to transition from an idle state (e.g., C6) to an active state (e.g., C0) and vice versa and includes steps like saving/restoring the context of the core in an external SRAM, switching on/off the VR or PLL, and flushing L1D/L2. All C-states incur a transition latency; the shallower the state, the shorter its transition latency. Usually, the latency reported by the vendors (see Table 1) along with each C-state represents the hardware transition round trip time and includes the time to transition to and from a C-state.

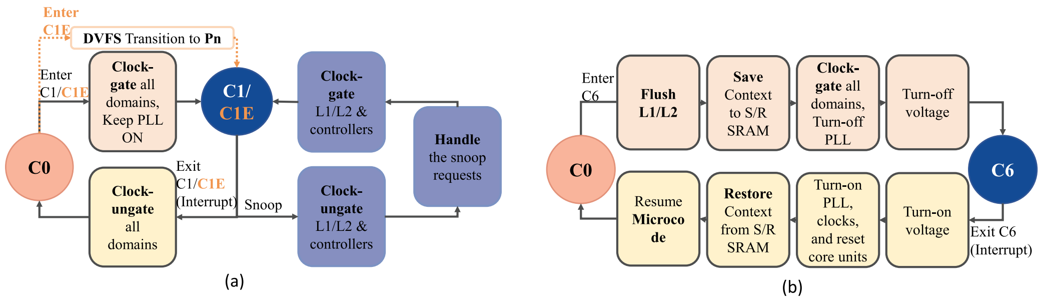
Enter C1E
DVFS Transition to Pn
Enter C1/C1E
Clock-gate all domains, Keep PLL ON
C1/ C1E
Clock-gate L1/L2 & controllers
Handle the snoop requests
C0
Clock-ungate all domains
Exit C1/C1E (Interrupt)
Snoop
Clock-ungate L1/L2 & controllers
(a)
Enter C6
Flush L1/L2
Save Context to S/R SRAM
Clock-gate all domains, Turn-off PLL
Turn-off voltage
C6
C0
Resume Microcode
Restore Context from S/R SRAM
Turn-on PLL, clocks, and reset core units
Turn-on voltage
Exit C6 (Interrupt)
(b)

Fig. 1. Entry and exit flows for C1 (a) and C6 (b).

Transition latency varies based on architecture and workload. Architecture choices, such as type of VR (i.e., Switching Voltage Regulator, Low Dropout Voltage Regulator [31, 34]) impacts core frequency changes (C1E, sets V/F of core to minimum), as the VR type affects the power conversion efficiency and thus the time it will take to ramp up the voltage and frequency to a target V/F. Workload also matters; cache "dirtiness" of a workload affects cache flush time (C6). We analyze the C6 entry/exit latency for a Skylake-based server architecture. The step that takes the longest to execute when entering C6 is flushing of the L1D/L2 cache. This duration is variable, determined by factors such as clock frequency and the count of dirty blocks in the cache. We conservatively choose a clock frequency of 800MHz and "cache dirtiness" of 50% to estimate the time of this step, which is around $\sim 75\mu s$. In comparison, the latency for transferring the core context (e.g., control registers, patch RAM) is $\sim 9\mu s$ for frequency of 800MHz and core context of 8KB. As a result, entry time of C6 including the flow overhead and the power-gate controller overhead is $\sim 87\mu s$. The exit time is dominated by the time required to restore the content of the core ($\sim 20\mu s$). Power ungating, PLL relock, and fuse propagation take $\sim 10\mu s$, making the overall exit time of C6 $\sim 30\mu s$.

2.2 Cold-Start Latency

Power gating is a technique used during deep core sleep states (C6). When a core is power gated, it is essentially turned off, causing the loss of an application's architectural and microarchitectural state (e.g., cache and branch predictor state). When reactivated to re-execute the same application on the core, it faces longer execution times, as the core needs to retrain its cache and branch predictor state, resulting in additional cache misses and branch mispredictions. The magnitude of this effect, which we call *cold-start latency*, depends on the amount of state that is lost when power gating and the performance benefit that the running application is able to extract from this state.

Previous work found cold-start latency to be in the order of microseconds [4, 10], with negligible impact on applications that execute for several milliseconds. However, as discussed earlier, microservice-based applications have per service active times of 30–250μs [10, 72, 97], making the impact of cold-start latency non-negligible.

2.3 Interplay of Core C-State Overhead with Core Components

In this section, we present how different core components affect the transition latency and cold-start latency of deep C-states. Specifically, we present how the state loss caused by power gating each core component can have consequences either on the performance or the correct functionality of the core and how these consequences correlate with the attributes (i.e., transition latency and cold-start latency) of C-states. The taxonomy plays a vital role in shaping the development of the new Agile C-state later on.

Table 3. Transition Latency and Cold-Start Latency Resources

Categories		**Transition Latency**	**Cold-Start Latency**
Architectural	**Caches**	**L1D, L2, DTLB, STLB**	L1I, Uop$, **L1D**, **L2**, ITLB, **DTLB**, **STLB**
	Context	Control Registers, Patch RAM, microcode	
Non-Architectural			Branch predictor, BTB, RAS, IJB, Prefetcher
PDN, Clock		PLL, FIVR, CLOCK	

Resources in bold belongs to both transition latency and cold-start latency.

In the taxonomy presented in Table 3, architectural resources represent the resources for which state loss (after power gating) can lead to incorrect core behavior. Patch RAM [57] is a representative example of this group. Patch RAM is memory within the core that holds updates to software drivers and firmware with the aim to protect the core against vulnerabilities that are identified after the core release. The state of the Patch RAM needs to be present once the core exits power gating for the core to have reliable behavior. Non-architectural resources represent resources for which state loss cannot lead to incorrect core behavior but rather affect the performance of the core. Branch predictor state is not necessary for the core to execute but can significantly optimize the execution time of a predictable program.

We observe that all of the resources that impact the transition latency are also architectural and all of the resources that affect cold-start latency are non-architectural. Finally, there is a subset of resources that affect both categories. L1D, for example, affects both transition and cold-start latency. Before power gating, the cache content needs to be saved, and after power ungating, the cache needs time to warm up. We include PDN and clock resources to complete the taxonomy.

The division of the resources into the groups we described previously allowed the division of the core structures into structures of which the state must be preserved by the proposed C-state (i.e., architectural) and structures of which the state must be preserved only if it significantly affects the performance of the core (i.e., non-architectural). It also offered an initial latency breakdown since the size of the state of the structure positively correlates with the overhead the structure causes.

2.4 Microservice Interplay with Core C-States

A microservice-based application is structured as a composition of multiple services that can be deployed independently. These services interact using well-defined APIs and communication protocols, such as HTTP, Message Queues, and RPC. Notably, a significant portion of the overall response time is attributed to communication overhead. Consequently, the computational aspects of microservices must adhere to stricter QoS constraints to meet the same requirements as compared to traditional monolithic applications. Usually, the QoS constraints per service are in the order of 30–250μs [10, 72, 97], making both the transition and cold-start latency notable.

Besides having short execution times, microservices operate at low utilizations (5%–25% [64, 65]) to keep the tail latency under control. However, as requests are characterized with irregular arrival times [20], the system's idle time is unpredictable. To avoid the transition and cold-start latency overheads, datacenter operators eventually disable the deep idle core states [3, 51, 92], even though a system that runs microservices is idle most of the time [64, 65]. Even if the request arrival time was predictable, and energy could be reduced by entering a deep idle core state, the cold-start latency problem remains. As a result, legacy core C-states are unattractive for modern servers running latency-critical microservice-based applications.

Finally, throughout its lifetime, a query executes on multiple services, each of which could run on different cores. Each core has its own idle governor who decides which C-state the core should enter. The primary heuristic used in the C-state selection algorithm is the local core history.

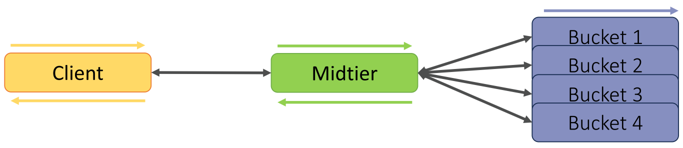

Fig. 2. Software architecture of MicroSuite benchmarks.

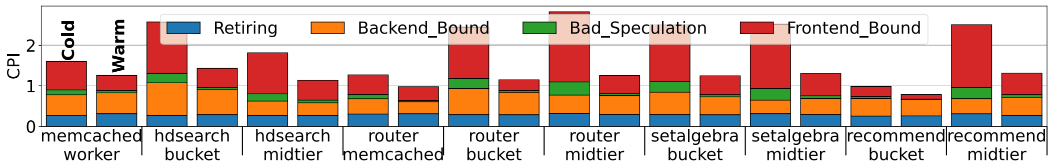

Fig. 3. Level 1 top-down analysis for Memcached and MicroSuite.

Consequently, due to the independent nature of the decision-making process for entering core C-states, a query may encounter cold-start latency multiple times during its execution.

2.5 Quantifying Cold-Start Latency

To estimate the worst-case impact of cold-start latency, we execute several microservice-based benchmarks. Memcached [21] is a lightweight key-value store that is widely deployed as a distributed caching service to accelerate latency-critical applications [93]. MicroSuite [87] is a microservice-based benchmark suite that contains information-retrieval services (HDSearch, Router, SetAlgebra, Recommend). *HDSearch* is an image similarity search service. *Router* is a Memcached protocol router. *SetAlgebra* performs posting list set intersection. Finally, *Recommend* is a recommendation service. Figure 2 presents the architecture of the MicroSuite benchmarks. To achieve the effect of cold and warm resources, we configure the C-states of the system to enabled or disabled. C-states enabled represents the scenario where a query is served on cold resources, since when the system enters C6, it power gates the core, and C-states disabled represents the scenario where a query is served with warm resources. To quantify the worst-case cold-start overhead, which represents a scenario where every query sent by the client executes on a cold core that exits C6 in response to the arrival of the query, we develop a new client. We configure the client to run on the same node as the benchmarks and have the following characteristics: (1) contains a warm-up phase to allow the system to enter C6 (system does not enter C6 for the first queries), and (2) sends queries with big enough interval that allows the system to enter C6 between successive queries.

We use Top-Down analysis [94] to examine the components of the core that contribute to the cold-start latency. Figure 3 presents the Level 1 Top-Down analysis. The analysis reveals that the worst-case impact of cold-start latency on the CPI of each tier of an application is significant and it ranges from 25% to 126%. A possible reason for this variation is the execution time at each tier.

Additionally, the analysis shown in Figure 3 reveals that the categories with the highest overhead are frontend bound and bad speculation. This is because, for both the frontend bound category and the bad speculation category, the contribution to the CPI increases when the application is running on cold resources versus when it is running on warm resources. The performance degradation caused by the frontend resources and bad speculation is 27%, 66%/55%, 21%/106%/126%, 92%/98%, and 21%/96% for Memcached, HDSearch, Router, SetAlgebra, and Recommend, respectively. In most of the cases examined, the midtier experiences higher cold-start overhead (on average ~93.75%) than bucket (on average ~71.25%). This is because midtier enters two times more the core C-state C6 during the lifespan of a query compared to a bucket: (1) when it waits for the client to send a query and (2) when it waits for bucket response.

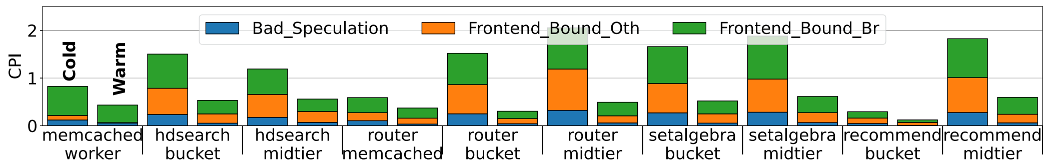

Fig. 4. CPI of frontend bound, bad speculation, and branch resteers categories for all benchmarks.

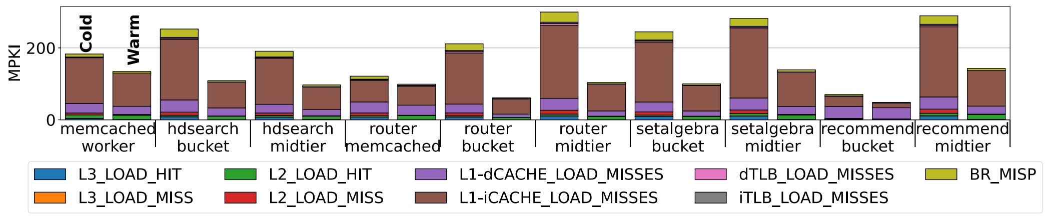

Fig. 5. MPKI for all benchmarks and tiers.

Figure 4 narrows down the sources of performance degradation to frontend bound–branch resteers, frontend bound–others (ITLB, L1I, etc.) and bad speculation. Specifically, it depicts the slowdown in terms of CPI for frontend bound and bad speculation over the baseline scenario (cold vs warm). We include branch resteers as it is the frontend category (Top-Down level 3) that contributes the most to the overhead caused by cold-start latency followed by ITLB misses and L1I misses. Branch resteers represent the number of slots lost between resolving the branch instruction until renaming an instruction from the correct path. Along with bad speculation, they represent the penalty caused by branch mispredicts. With cold resources, the penalty caused by branch mispredicts as compared to warm resources increases by 46%–233% (~118% on average). The overhead caused by the rest of the frontend resources increases by 12%–166% (~50% on average).

Figure 5 presents the MPKI instruction analysis for all benchmarks and tiers evaluated. It is interesting to notice that the MPKI figure correlates with the results of the Top-Down analysis since the instruction cache and the branch predictor are two of the structures with the highest number of misses. Both L1D and L1I cache have high MPKI for both the cold and warm scenarios, indicating a large instruction footprint and working set for all tested benchmarks.

We have shown that the cold-start overhead affects the performance of a microservice . The categories with the highest impact are frontend bound and bad speculation. Out of all resources of the frontend, branch predictors cause the highest overhead followed by L1I cache and iTLB. We conclude that all data arrays of the core (both architectural and non-architectural) contribute significantly to the cold-start overhead and need to be kept warm in a C-state architecture targeted for microservice-based latency-critical applications.

2.6 Real-World Impact of Cold-Start Latency

The dynamic workload behavior of latency-critical applications based on microservices prevents cores from entering deep sleep states even during periods of low utilization [5, 11, 51, 64]. As a result, in a real-world scenario, a workload would rarely experience cold-start latency. Thus, in this section, we analyze whether or not it is important to consider cold-start latency in the design of the new Agile C-state architecture, and what the impact of cold-start latency on the performance of a microservice would be if we only optimized the transition latency of C6. We do this by estimating the worst-case impact of cold-start latency on the average response time for a C-state with transition latency the same as C1 and cold-start latency the same as C6. Since microservices

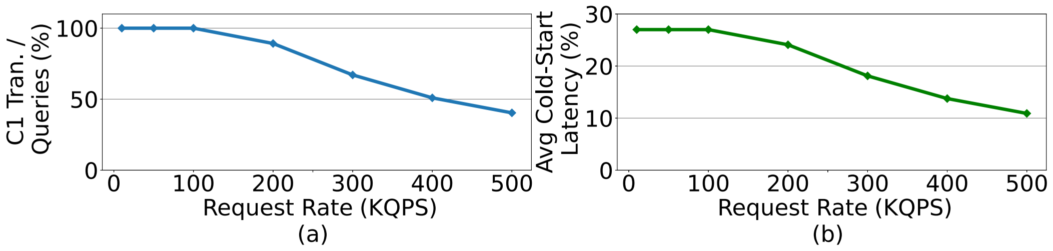

Fig. 6. (a) Queries that enter C1 for Memcached. (b) Cold-start latency impact on average response time.

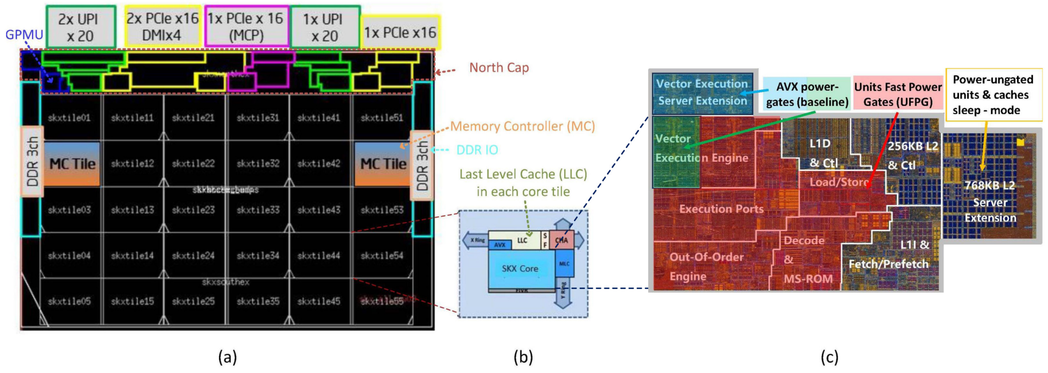

Fig. 7. Skylake server (SKX) architecture. (a) SKX floorplan [3, 88]. (b) SKX core tile [3, 88]. (c) SKX Core + C6Awarm architecture [55, 92].

operate at low utilization to keep the tail latency under control, we consider utilization between 5%–25% [64, 65] normal conditions, but we also examine higher utilization.

Figure 6(a) presents an upper limit of the percentage of Memcached queries that will enter C1 and thus experience cold-start latency. This is an upper limit because we do not take into account how the execution time increase caused by cold-start overheads affects the idle time of the system and thus the number of C1 transitions. Figure 6(b) presents how much the average response time of Memcached will increase due to cold-start latency based on the number of C1 transitions. Our analysis shows that for query rates between 10K and 100K QPS (queries per second) (5%–15%) all queries of Memcached can experience cold-start overheads and so the average response time increases by 27%. For the higher QPS of 200–500K (20%–60%), 90% to 40% of queries experience cold-start latency—in other words, the average response time increases from 24% to 10.8%. The rest of the article illustrates how C6Awarm/C6AwarmE can eliminate both transition and cold-start latency while maintaining most of the power savings of deep idle states (i.e., C6).

3 AgileWatts Architecture

Before presenting **AgileWatts (AW)**, we first describe the characteristics of the architecture for a Skylake-based processor. Figure 7(a) presents the floorplan of a Skylake-based server processor (SKX). The processor's main components are the core tiles, the on-die-fabric, the north cap [88, 90, 91], and the memory controllers. The on-die-fabric, which is organized as a mesh network, connects all main components of the architecture. The core tiles, which cover most of the chip's area, contain all resources of the core domain including some resources of the uncore domain (see Figure 7(b)). Specifically, the core domain contains the CPU core, AVX extension, and private caches (see Figure 7(c)), and the uncore domain of the core tile contains the caching-and-home-agent (CHA), last-level-cache (LLC), and snoop filter (SF). The north cap, which is the area located at the top of the die, consists of the IO agents (i.e., PCIe, UPI, and DMI), the serial ports, the clock generator unit, the fuse units, and the global power management unit. Finally, the die contains two memory controllers on the left and right columns. Since AW optimizes the core C-state architecture, we emphasize on the differences between Skylake and AW on the core domain (see Figure 7(c)).

This work extends the initial version of AW [92] (i.e., C6A/C6AE) and introduces two new core deep idle power states: C6Awarm and C6AwarmE, which aim to optimize both the transition and cold-start latency while achieving significant power savings. C6A and C6AE (i.e., initial version [92]) optimize only the transition latency of deep idle power states. Additionally, in the initial version of AW, the L1I & Fetch/Prefetch chip area (see Figure 7(c)) is power gated, whereas in C6Awarm is kept into retention mode.

C6Awarm (C6A Agile warm) has close to zero-watt power consumption and nanosecond-scale entry/exit latency which allows servers running latency-critical applications to enter it during short and irregular idle periods. C6AwarmE (C6 Agile warm Enhanced, analogous to C1E) is a lower-power variant of C6Awarm that reduces additional leakage power by lowering core voltage to a minimum operational level. The discussion focuses on the C6Awarm design and operation and points out the differences with C6AwarmE when relevant.

The C6Awarm state is based on two key ideas: **Units' Fast Power Gating (UFPG)** (Section 3.1) and a **Cache Coherence and Sleep Mode (CCSM)** (Section 3.2). The new power management flow that coordinates the UFPG and the CCSM at nanosecond granularity is presented in Section 3.3.

3.1 Units' Fast Power Gating

AW UFPG is a low-latency power gating architecture that shuts off most of the core units while retaining the context in place, thus enabling a transition latency of tens of nanoseconds. Conventional context retention techniques (see C6 C-state flow in Figure 1(b)) sequentially save/restore the context to/from external SRAM before/after power gating/ungating [19, 33, 35, 79]. This process adds several microseconds (e.g., 5–10µs) to the entry/exit latency. Instead, AW retains the context in place, completely removing that overhead at a very small additional idle power cost.

AW enables in-place context retention with a *medium-grained* power gating approach. In *coarse-grained* PG (used in Skylake client cores), the entire core is under the same power gate [19, 38, 47, 81] and context is saved/restored externally. Our approach is based on the power gating for the AVX-256 and AVX-512 core units [19, 33, 68] in recent server and client cores. These power gating techniques require only 10–20ns to power gate/ungate a unit [19, 33] because they power and retain the unit's context in place and avoid having to save and restore it externally.

AW's *medium-grained* power gating for C6Awarm and C6AwarmE applies to the majority of the core units (shaded red in Figure 7(c)) and excludes the L1 and L2 caches (instruction and data), their controllers (handled separately in Section 3.2), and the branch predictors along with their controllers (see Figure 7(c), L1I Fetch/Prefetch).

Within the medium-grained PG region, AW uses multiple techniques to retain the context in place and enable fast (several nanoseconds) transition latency. The context of a CPU core is ~8kB[1] [25, 46] and falls into two categories: (i) *registers* (i.e., configuration and status registers (CSRs) or fuse registers) and (ii) *SRAMs* (i.e., firmware persistent data and patches) [35, 79]. We discuss

[1]Haj-Yahya et al. [35] show that the context saved/restored for an entire Skylake client mobile SoC is ~200KB. This includes the context of four cores, an integrated GPU, the uncore, and the system agent. We estimate the single-core context based on a core's relative die area, showing ~8kB context, similar to previous Intel references [25, 46].

next three techniques that AW uses to retain the context during C6Awarm; the first two apply to registers and the third to SRAM.

Placing Unit Context in the Ungated Domain. One option to retain the context of a power-gated unit is to place its registers outside the power-gated region (i.e., in the core's ungated domain). This option is suitable for units with small context (e.g., execution units); Intel likely uses this technique for the AVX execution units [58, 68]. AW uses this technique for all core units that require only a local context to be retained—that is, this is not applicable to a unit with a distributed context that is impractical to relocate to a centralized ungated region. The following units satisfy this requirement: (1) all execution units (not AVX), (2) execution ports, and (3) the out-of-order engine.

State Retention Power Gates. Moving distributed or large context to a separate ungated area is impractical (e.g., timing and wiring constraints). For this reason, AW employs a different retention technique—**State Retention Power Gates (SRPGs)**—for units that contain such context. SRPG is a special flip-flop fed with two supplies: power gated and power ungated. Such a flip-flop typically contains a shadow flip-flop to retain its state when the unit it resides in is power gated [58, 67, 75]. Intel uses this technique in the chipset to retain the state of autonomously power-gated units [68].

Place SRAM Context in Ungated Power Supply. Part of the CPU core context is located in SRAMs [35]. While the microcode firmware is stored in read-only memory, known as microcode sequencer ROM (MS-ROM), microcode patches and data are stored in an ~2KB SRAM [18, 29]. This SRAM is initialized at boot time and should be retained when power gating the microcode unit. The C6 exit flow re-initializes the content of this SRAM from the core's S/R SRAM in a separate ungated uncore domain; this process is sequential and can take several microseconds [35, 79]. AW avoids the need to re-initialize the microcode patch SRAM by powering it with a separate core ungated supply.

3.2 Cache Coherence and Sleep Mode

To avoid the high transition latency (tens of microseconds) to flush private caches (i.e., L1D and L2) and the high cold-start latency (see Section 2) to warm up the microarchitectural state of the core (branch predictors, L1I ~ L1I & Fetch/Prefetch, see Figure 7(c)), AW keeps them power ungated (see Figure 7(c)) when transitioning to C6Awarm. This has two design implications: first, AW needs to employ other power-saving techniques to reduce the power of the cache/L1I & Fetch/Prefetch domain, and second, a core in C6Awarm state still needs to serve coherence requests [30, 70].

AW employs two techniques to reduce the power consumption of the power-ungated private cache/L1I & Fetch/Prefetch domain. First, unless a coherence request is being served, AW keeps these domains clock gated to save its dynamic power. Second, AW leverages the *cache sleep mode* technique [9, 22, 40, 82], which adds sleep transistors to the SRAM arrays of private caches and branch predictors. The sleep transistors reduce the SRAM array's supply voltage to the lowest level that can retain the SRAM content while significantly reducing leakage power.

Since private caches are not flushed when a core enters C6Awarm, AW must allow the core to respond to snoop requests [26, 32, 81]. AW keeps the logic required to handle cache snoops in the power-ungated (clock-gated) domain with the private caches. It also uses minimal logic (*same with C1*) to detect incoming snoop requests in an always-active (i.e., not power/clock-gated) domain. As soon as this logic detects incoming snoop traffic, it increases the SRAM array voltage through sleep transistors and re-activates the clock of the private caches until the snoop requests are served.

3.3 AW Power Management Flow

AW implements the C6Awarm/C6AwarmE flow within the core **power management agent (PMA)** [80]. This flow, shown in Figure 8, orchestrates the transitioning between the C0 and

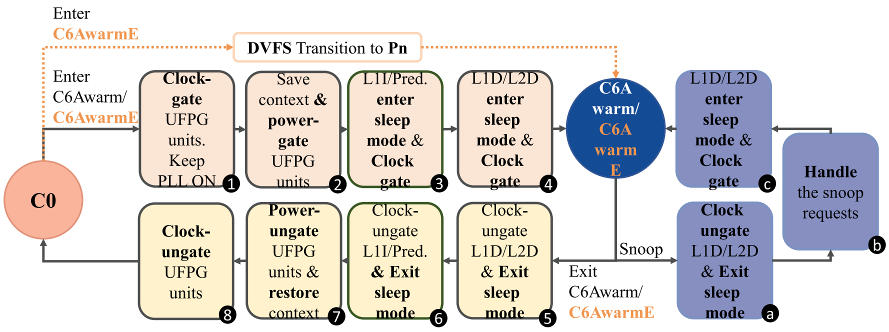
Enter C6AwarmE
DVFS Transition to Pn
Enter C6Awarm/ C6AwarmE
Clock-gate UFPG units. Keep PLL ON 1
Save context & power-gate UFPG units 2
L1I/Pred. enter sleep mode & Clock gate 3
L1D/L2D enter sleep mode & Clock gate 4
C6A warm/ C6A warm E
L1D/L2D enter sleep mode & Clock gate c
Handle the snoop requests b
Clock ungate L1D/L2D & Exit sleep mode a
Snoop
Exit C6Awarm/ C6AwarmE
Clock-ungate L1D/L2D & Exit sleep mode 5
Clock-ungate L1I/Pred. & Exit sleep mode 6
Power-ungate UFPG units & restore context 7
Clock-ungate UFPG units 8
C0

Fig. 8. Power management flows for the C6Awarm/C6AwarmE states.

C6Awarm/C6AwarmE C-states and handles coherence traffic while in the C6Awarm/C6AwarmE state.

Similar to other C-states, the operating system triggers C6Awarm *entry* by executing the MWAIT instruction [26, 32]. The first step ❶ clock gates the UFPG domain (Section 3.1) and keeps the core **phase-locked loop (PLL)** on. When entering C6AwarmE, the PMA initiates a non-blocking transition to Pn. Subsequently ❷, the flow saves (in place) the UFPG domain context and shuts down its power. Additionally ❸, the flow puts L1I cache and branch predictors into sleep mode and clock gates the domain. Finally ❹, the flow sets the private caches into *sleep mode* (Section 3.2) and shuts down their clock, putting the core into *C6Awarm/C6AwarmE* state.

When a *snoop* request arrives, the PMA activates the private caches to respond. First, ⓐ the flow clock ungates the cache domain and adjusts its supply voltage to exit *sleep mode*. At this point ⓑ, the caches can handle the snoop requests. Finally ⓒ, PMA rolls back the changes in reverse order and brings the core back into *C6Awarm/C6AwarmE* state once all snoops are serviced.

When an *interrupt* occurs, the core *exits* from *C6Awarm/C6AwarmE* and goes back into *C0* (active) state. The exit flow is simply the reverse process of the entry flow. First ❺, the flow clock ungates L1/L2 and exits *sleep mode*. The flow also ❻ clock ungates the L1I–branch predictor and exits sleep mode. Next ❼, it power ungates the UFPG units and triggers the restore signal to the SRPG flops. Finally ❽, the flow clock ungates UFPG units, bringing the core to the *C0* active state.

4 AW Implementation and Hardware Cost

As discussed in Section 3, AW requires the following in each CPU core: (1) the UFPG subsystem, (2) the CCSM subsystem, and (3) the C6Awarm controller. This section discusses the implementation of each component, its power-performance-area cost, and the resulting transition latency for the new C6Awarm and C6AwarmE states.

4.1 Power-Performance-Area Modeling Methodology

In this section, we describe each of the modeling components in detail. Table 4 summarizes the total area overhead and power consumption of AW's C6Awarm/C6AwarmE C-states. Our power and performance model (described in Section 5.1) uses C6Awarm/C6AwarmE power and AW performance overheads to estimate the average power consumption and performance impact of AW for a given workload. Our discussion focuses on the Power, Performance, and Area overhead of C6Awarm and points out the differences of C6AwarmE when relevant.

4.1.1 Units' Fast Power Gating. As discussed in Section 3.1, AW's UFPG places the majority of the core units behind power gates that are similar to the ones used for the AVX units in recent Intel cores. AW uses power gates for ~55% of the core area (measured on the die photo in Figure 7(c)).

Table 4. Area and Power Requirements to Implement AW in a Skylake-Like Core

Component	Sub-Component	Area Requirement	C6Awarm Power	C6AwarmE Power
Units' Fast Power Gating (UFPG)	Units' power gates (~70% of the core)	2%–6% of power-gated area	~24–40mW$^{\alpha}$	~15–24mW^{α}
	Ungated context registers	<1% of ungated context registers		
	State retention power gates (SRPG)	<1% of gated unit area	~2mW$^{\beta}$	~1mW$^{\beta}$
	Ungated context SRAM	<1% of SRAM area		
Cache Coherence & Sleep Mode (CCSM)	L1I/L1D/L2/branch predictors sleep mode	2%–6% of private cache, LI1 fetch/prefetch area	162mW$^{\delta}$	111mW$^{\gamma\ \delta}$
	Rest of ungated memory subsystem	<1% of the ungated units		
PMA Flow	Implemented in the uncore [80]	<5% of core PMA [35]	5mW$^{\epsilon}$	5mW$^{\epsilon}$
Core ADPLL & FIVR	ADPLL	0%	7mW [19]	7mW [19]
	Core FIVR inefficiency	0%	36–41mW$^{\zeta}$	23–27mW$^{\zeta}$
	FIVR static losses	0%	100mW$^{\eta}$	100mW$^{\eta}$
Overall		3%–7% of the core area	336–357mW	262–275mW

$^{\alpha}$Assuming 3%–5% [23] of leakage power ≈ C1 power. $^{\beta}$Power of the ~8KB context [35]. $^{\gamma}$Higher sleep transistor efficiency at V_{min} (C6AwarmE) [31, 39, 66]. $^{\delta}$Extrapolated from a similar implementation of prior work [92]. $^{\epsilon}$Based on scaled wake-up logic power from prior work [35]. $^{\zeta}$Assuming 80% FIVR efficiency in light load [31, 34, 76]. $^{\eta}$FIVR static losses [5, 31, 34] in C6 state are ~100mW [71].

Power Overhead. The AVX power gates and the new UFPG shut off all units in the core execution domains; however, power gates only eliminate 95%–97% of the leakage power [23, 49], thus the UFPG domain has residual idle power while in C6Awarm. Using the Intel core-power-breakdown tool [36], we derive the leakage power contribution of the power-gated units starting from the leakage power of the entire core.[2] Our estimation shows that the C6Awarm power-gated area contributes to approximately 55% of the core leakage (i.e., ~55% of $C1$ power). Hence, the power overhead of UFPG (i.e., 3%–5% of the gated leakage power) is ~24–40mW at base frequency (P1, C6Awarm) or ~15–24mW at minimum frequency (Pn, C6AwarmE).

The three UFPG techniques combined retain ~8KB context [18, 29], which consume ~0.2mW at retention voltage [35]. To estimate the retention power at base (P1) and minimum (Pn) frequencies, we conservatively multiply the retention-level power by 10× and 5×, respectively. Our estimate for context retention power is ~2mW (P1, C6Awarm) to ~1mW (Pn, C6AwarmE).

Performance Overhead. In an active CPU core, simultaneous operations in memory or/and logic circuits demand high current flow, which creates fast transient voltage droops [27, 28]. One power-gating design challenge is the resistive voltage (IR) drop across a power gate, which exacerbates voltage droops [31, 34, 52, 76]. The worst-case voltage droop can limit the maximum attainable frequency at a given voltage since it requires additional voltage (droop) margin above the nominal voltage to enable the CPU core to run at the target frequency [76]. An x86 implementation of a CPU core power gate leads to <1% frequency loss [52]. Our AW analytical model (Section 5.1) assumes 1% *frequency degradation* due to UFPG.

Area Overhead. A power gate adds 2%–6% extra area to the gated logic (i.e., *UFPG*, covering ~55% of core area). We conservatively use the wide overhead range (i.e., 2%–6%) since the exact overhead depends on the implementation, the size of the gated area, the number of required isolation cells,[3] and technology [23, 99]. The area overhead for the in-place *context retention* techniques

[2]The leakage of the entire core is approximately equal to the $C1$ power (see Table 1), which removes only dynamic power by applying clock gating.

[3]Isolation cells isolate the always-on units from the floating values of the power-gated units. They are typically placed on the outputs of the power-gated domains during the physical placement stage [8].

of the ~8kB core context [25, 46] is as follows. First, moving the context of a unit to ungated power typically requires <1% of the context area, mainly due to the isolation cells [8]. Second, the use of SRPGs for components with too large or distributed context is a mature technique already used in productssuch as the Intel Skylake [68]. Efficient SRPG designs, which use *selective* context retention, require less than 1% additional area relative to the power-gated area they control [75]. Finally, including an SRAM into the ungated domain requires isolation cells that add overhead <1% of the SRAM area [8].

4.1.2 Cache Coherence and Sleep Mode. AW implements sleep mode for private caches similarly to the sleep mode used for L3 caches in multiple server products [9, 40, 82]. Cache sleep mode implements P-type sleep transistors [9, 40] with seven programmable settings and local bit-line float to reduce SRAM cell leakage in the data array; it also employs word-line sleep to reduce leakage further.

Power Overhead. *CCSM* implements sleep mode using sleep transistors in the L1I/L1D/L2 SRAM data array and branch predictors; this technique is already used for efficient design of the L3 cache in multiple server processors on the market [9, 40, 82]. Based on a comparable CCSM implementation [92], we extrapolate a retention factor (0.25), representing how much the leakage power of a core array will reduce once we apply sleep transistors on it. We do not use the same methodology as in prior work [92] for estimating the power consumption of CCSM, since characteristics of structures included in the CCSM domain of C6Awarm, such as size of the branch predictor, are unknown. However, we try to validate the results of the methodology described previously using the same methodology as in prior work [92] and pessimistic structure sizes. The resulting leakage power estimation for L1/L2 and branch predictors is 162mW for C6Awarm and 111mW for C6AwarmE, respectively.

Performance Overhead. In AW, only the data array (accounts for more than 90% of L1/L2 cache size) is placed in sleep mode, whereas the other control arrays (e.g., tag, state) operate at nominal voltage. Doing so allows *hiding the data array wake-up latency* during the control array access time, thereby eliminating any performance degradation compared to operation without the sleep mode.

Area Overhead. Implementing sleep mode using sleep transistors for the SRAM data array of the private caches and branch predictors requires additional area similar to power gates (i.e., 2%–6% of the SRAM area) [22, 23]); a recent implementation reports a 2% area overhead [99].

4.1.3 AW Power Management Control Flow. The main implementation consideration to realize the C6Awarm flow in Figure 8 is a mechanism to control in-rush current [19, 48]. This needs to support staggered wake-up, so as to ensure PDN stability [1, 19, 33, 48, 53]. The remaining capabilities, such as clock gating and event detection (interrupts, snoops), are all commonly supported in state-of-the-art SoCs. The C6Awarm controller is implemented using a simple finite-state machine within the core's PMA, which resides in the uncore [80] and controls clock gating/ungating, save/restore signals, and L1I/L1D/L2/branch predictor entry-to/exit-from sleep mode. The *C6Awarm* snoop flow reuses the existing snoop handling mechanisms of the *C*1 state (shown in Figure 1).

Power Overhead. AW implements the C6Awarm controller as finite-state machine within the PMA. Based on a comparable power management flow implemented in prior work [35], we estimate that the C6Awarm controller adds approximately 5mW to the PMA power.

Performance Overhead. The additional control circuit we add to the PMA has no direct impact on the CPU core's performance. The performance overhead is mainly due to the new features the PMA controls (described in Section 4.1.1 and Section 4.1.2).

Area Overhead. Based on a comparable power management flow implemented in prior work [35], we estimate the additional area to implement the C6Awarm controller to be up to 5% of the core PMA area.

4.1.4 Core PLL and FIVR. AW keeps the PLL and FIVR powered on in C6Awarm/C6AwarmE states. Next we describe only the power overhead of the PLL and FIVR as there is no extra performance or area overhead compared to baseline.

Power Overhead. We estimate the C6Awarm idle power needed by AW to keep the PLL on and locked while accounting for voltage regulator inefficiencies. The Skylake core uses an ADPLL and a FIVR [19, 91]. The ADPLL consumes 7mW (fixed across core voltage/frequency levels [19]). The FIVR presents dynamic efficiency loss due to conduction and switching inefficiency [59], and static efficiency loss due to power consumption of the control and feedback circuits [31, 59, 71]. The static loss still applies when the FIVR output is 0V. The FIVR static loss accounts for ~100mW per core [5, 31, 34]. The FIVR efficiency at light load is about 80% (excluding the static power losses) [31, 34, 76].

4.2 C6Awarm and C6AwarmE Latency

We estimate the transition time (i.e., entry followed by direct exit) required by the extra actions taken by C6Awarm/C6AwarmE of AW compared to C1/C1E to be <90ns. This is three orders of magnitude faster than the >100µs latency of C6. Thus, the overall transition time of C6Awarm/C6AwarmE is 90ns plus the transition time of C1. Next, we explain in detail this estimation using Figure 8.

C6Awarm and C6AwarmE Entry Latency. Clock gating all domains and keeping the PLL ON (❶ in Figure 8) typically takes 1–2 cycles in an optimized clock distribution system [17]. Transitioning to *Pn* (required for *C6AwarmE*) happens with a non-blocking parallel DVFS (i.e., P-state) flow that can take few tens of microseconds, depending on the power management architecture [24]. Since AW keeps the context in-place, saving the context to power gate the core units ❷ only requires asserting the `Ret` signal followed by de-asserting the `Pwr` signal . We estimate this process to take 3–4 cycles. Finally, placing the L1I/L1D/L2 caches and branch predictors in sleep mode and clock gating them ❸ takes 1–3 cycles. Hence, the overall entry flow takes <10 cycles—that is, <20ns with a power management controller clocked at 500MHz. [4]

C6Awarm and C6AwarmE Exit Latency. Clock ungating the L1I/L1D/L2 caches and branch predictors and exiting sleep mode ❹ takes 2 cycles [40]. Power ungating the core units ❺ takes <60ns, and restoring the core context (i.e., de-asserting the `Ret` signal after restoring power) takes 1 cycle. Finally, clock ungating all domains ❻ typically takes 1–2 cycles. Hence, the overall exit flow takes ~5 clock cycles + <60ns, equivalent to <70ns when using a 500MHz clock for C6Awarm/C6AwarmE.

C6Awarm and C6AwarmE Snoop Handling. Snoop handling latency in C6Awarm (C6AwarmE) is similar to that in C1 (C1E). Specifically, clock ungating the L1I/L1D/L2 caches and exiting sleep mode ⓐ takes 2 cycles. In the first cycle, the flow ungates the clock; in the second cycle, the snoop requests simultaneously (1) access the cache tags (power ungated) and (2) wake up the cache data array [9, 40, 82]. Placing the L1/L2 in sleep mode and clock gating L1/L2 caches ⓒ after servicing the snoop traffic ⓑ takes 1–3 cycles.

5 Experimental Methodology

We evaluate AW using Memcached [21] and MicroSuite [87] (see Section 2.5).

[4]Typically, a power management controller of a modern SoC operates at a frequency of several hundred megahertz (e.g., 500MHz [74]) to handle nanosecond-scale events, such as *di/dt* noise [19; 33, Section 5].

System. We use the CloudLab [16] infrastructure for the experimental evaluation. Our baseline server is equipped with two Intel Xeon Silver 4114 [43] Skylake-based processors running at a nominal frequency of 2.2GHz (minimum frequency of 0.8GHz and maximum Turbo Boost frequency of 3GHz), with 10 physical cores for a total of 20 hyper-threads, and with 192GB DDR4 DRAM. We choose to disable turbo in all cases to eliminate variations among runs.

Workload Setup. For evaluating Memcached, we run a single Memcached server process on one of the server machines and run a modified version of the Mutilate load generator [61] on the rest of the machines. We configure the load generator to recreate the ETC workload from Facebook [6], using one master and four workload generator clients, each running on a separate server machine. We use three nodes to evaluate MicroSuite benchmarks (client, midtier, bucket). Our benchmark configuration follows the configuration of the MicroSuite work [87]. Finally, we pin the processes onto specific cores to eliminate process migration.

5.1 Power and Performance Model

We model the average power of a core using the fraction of time spent at each unique C-state and its corresponding power consumption. Similar to prior works [5, 10, 11, 54, 64, 69, 97], we focus on CPU power, which is the single largest contributor to server power [50, 89]. Next, we describe our models for the baseline and AW.

Baseline CPU Core Power. Our analytical power model estimates the average CPU core power for a workload, assuming that P-states/Turbo are disabled, as $Avg_P = \sum_{i \in \{0,1,1E,6\}} P_{C_i} \times R_{C_i}$. P_{C_i} denotes the core power in state Ci (reported in Table 1). R_{C_i} denotes the residency at Ci—that is, the percentage of the total time the system spends at state Ci. We obtain C-state residency and number of transitions using the processor's residency reporting counters [42]. When executing our workloads, we use the RAPL interface [41] to measure power consumption. The power model used for the baseline scenario has been validated and has an accuracy of 95% [92].

AW CPU Core Power. We model the power consumption of the CPU core enhanced with the two new C-states of AW (i.e., $C6Awarm$ and $C6AwarmE$) using (1) measured data from our baseline power model (C-state residency is scaled using our performance model; more details are presented in the following), and (2) estimated power of the $C6Awarm$ and $C6AwarmE$, as summarized in Table 4. We compute the average power in a similar way as for the baseline core, summing the power of each C-state weighted by its residency fraction.

To estimate power for the new C-states, we perform the following steps. First, we obtain the power and residency of each core C-state from the baseline. We scale the C-state residency taking into account (1) how the small core frequency degradation incurred due to the power gates (Section 4.1.1) affects performance by considering a workload's frequency scalability[5] and (2) the higher $C6Awarm/C6AwarmE$ transition latency (i.e., 90ns; Section 4.2) compared to $C1/C1E$. Second, we replace $C1/C1E$ C-state residency (i.e., $R_{C_1}/R_{C_{1E}}$) with $C6Awarm/C6AwarmE$ C-state residency (i.e., $R_{C_{6Awarm}}/R_{C_{6AwarmE}}$). Third, we replace $C1/C1E$ power consumption (i.e., $P_{C_1}/P_{C_{1E}}$) with $C6Awarm/C6AwarmE$ *estimated* power consumption (i.e., $P_{C_{6Awarm}}/P_{C_{6AwarmE}}$ in Table 4). We plug in the new values to estimate the average AW CPU core power.

We compute AW's average power for workloads with Turbo enabled (and P-state disabled) as $AvgP_{savings} = R_{C_1} \times (P_{C_1} - P_{C_{6Awarm}}) + R_{C_{1E}} \times (P_{C_{1E}} - P_{C_{6AwarmE}})$ and then compute savings relative to the baseline as $AvgP_{savings\%} = (AvgP_{savings}/AvgP_{baseline}) \times 100$, where we measure $AvgP_{baseline}$ using RAPL. Doing so allows our model to take into account power consumption variation during the $C0$ active state due to Turbo transitions.

[5]We define the frequency scalability of a workload as the change in its performance with unit change in frequency, as in other works [24, 37, 95, 96].

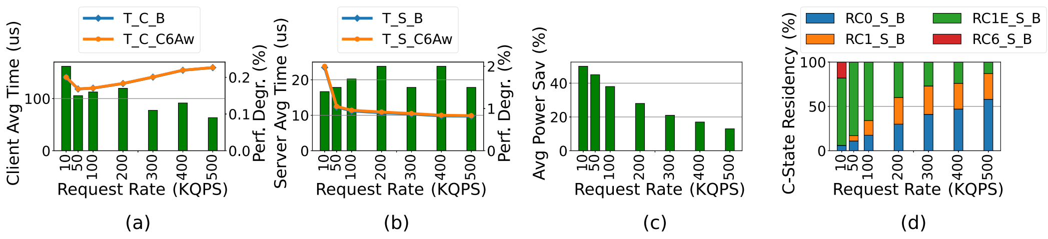

Fig. 9. Comparison of C6Awarm with baselines (P-state disabled, Turbo disabled, C-states enabled). (a) Client-side perf. degr. of C6Awarm. (b) Server-side perf. degr. of C6Awarm. (c) C6Awarm core average power savings when replacing *C*1 with *C6Awarm* and *C*1*E* with *C6AwarmE*. (d) Residency of baseline at different C-states. T_C_*, Client time baseline/C6Awarm; T_S_*, Server time baseline/C6Awarm; RC*_S_B, Residency Cx server-side baseline.

Naming Conventions. We use a triple-based naming scheme for concisely referring to different metrics and scenarios: *Metric_Process_Scenario*, where *Metric* can be *T*: Time (ms, us), *P*: Power (W), *Process* can be *M*: Midtier, *B*: Bucket, *C*: Client, *S*: Server, and *Scenario* can be *B*: Baseline *C6Aw*: C6Awarm.

Performance Model. We model the average response time for each benchmark on each scenario (Baseline, C6Awarm) using a combination of real measurements and equations. Specifically, we use the fraction of time each query spends on each tier (i.e., midtier, bucket, server), the number of C-state transitions, and the scalability analysis of each benchmark. We assume that the network/queueing time and the decisions of the idle governor remain the same.

Baseline CPU Core Performance. To estimate the performance impact of C6Awarm, we first calculate the performance of the baseline scenario experimentally (we use counters to extract the timing regions we are interested in). In our baseline run, we extract three performance values: the midtier processing time (*T_M_B*), the bucket processing time (*T_B_B*), and the end-to-end average response time (*T_C_B*). Since the tiers of the benchmarks evaluated execute sequentially, we are able to estimate the impact of C6Awarm on each tier and then use addition to evaluate the impact on the end-to-end response time.

AW CPU Core Performance. Once we have the baseline performance, we estimate the overhead of C6Awarm on each tier (midtier, bucket). We scale the midtier/bucket processing time by taking into account the frequency degradation due to power gating (see Section 4.1.1) using frequency scalability analysis and the higher transition latency (i.e., 90ns; Section 4.2) of C6Awarm/C6AwarmE. Then we add up the overheads of each tier and estimate the impact on the end-to-end average response time.

6 Evaluation

We first evaluate C6Awarm/C6AwarmE using Memcached in terms of performance degradation and power savings over a baseline scenario optimized for power savings (Turbo disabled/P-states disabled/C-states enabled). In Section 6.2, we evaluate C6Awarm using Memcached over a baseline scenario optimized for performance (Turbo disabled/P-states disabled/Deep C-states disabled). Finally, in Section 6.3, we present additional evaluation of C6Awarm with the MicroSuite benchmarks.

6.1 AgileWatts vs Power-Optimized Configuration

Figure 9 shows how AW affects request latency against the baseline with Turbo and P-states disabled (i.e., frequency is set to nominal frequency, *P*1: 2,200MHz) and C-states enabled. We expect

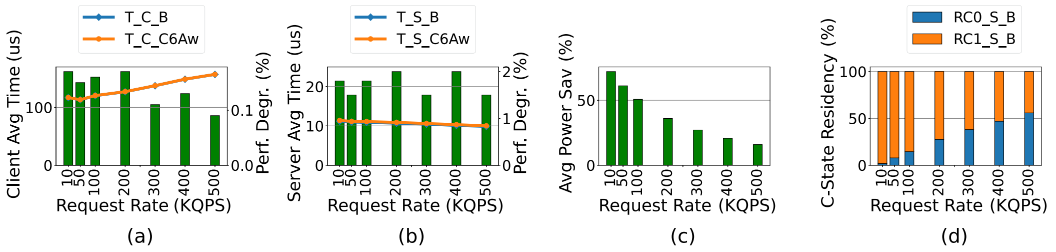

Fig. 10. Comparison of C6Awarm with baselines (P-state disabled, Turbo disabled, C-states disabled). (a) Client-side perf. degr. of C6Awarm. (b) Server-side perf. degr. of C6Awarm. (c) C6Awarm core average power savings when replacing *C*1 with *C6Awarm*. (d) Residency of baseline at different C-states. T_C_*, Client time baseline/C6Awarm; T_S_*, Server time baseline/C6Awarm; RC*_S_B, Residency Cx server-side baseline.

AW to achieve significant power savings, thanks to the lower power of *C6Awarm*/*C6AwarmE* compared to *C*1/*C*1*E*, while having a small impact on request latency because of the larger transition time (~90ns) and the ~1% frequency loss (Section 4.1.1). While the larger transition time impacts each C-state transition, the impact of the frequency loss depends on the sensitivity of the workload to the core frequency reduction.

Figure 9(d) shows the residency of the system in each C-state: as expected, idle time progressively reduces as load (QPS) increases. Therefore, we expect AW to have larger power savings and lower impact on performance at low load. Figure 9(c) shows that AW reduces the average power consumption by up to 50% at low load, with less than 2% impact on the server side and less than 0.2% impact on the client side (see Figure 9(a) and (b)). At high load, AW still provides 13% power savings, with less than 1% impact on the server side and 0.1% impact on the client side. Lo et al. [64] state the following in their work that discusses latency-critical applications [64]: "Modern servers are not energy proportional: they operate at *peak* energy efficiency when they are fully utilized, but have much lower efficiencies at lower utilizations." The utilization of servers running latency-critical applications is only 5%–25% to meet target tail latency requirements, as reported by multiple works from academia and industry [7, 64, 65]. For example, recently, Alibaba reported that the utilization of servers running latency-critical applications is typically 10% [65]. Therefore, our AW proposal addresses the more inefficient aspect of modern servers: running latency-critical microservice-based applications at low utilization.

We conclude that AW significantly reduces core average power consumption of the Memcached service across various load levels with minimal performance overhead over the baseline when disabling P-states and Turbo and enabling C-states.

6.2 AgileWatts vs Performance-Optimized Configuration

As discussed in Section 2.4, datacenter operators disable deep idle core states to prevent queries from incurring transition and cold-start latency overheads [3, 51, 92]. In this section, we evaluate AW benefit for servers at which deep C-states are disabled. Figure 10 shows how C6Awarm C-state affects the average response time and the power savings of the baseline (Turbo disabled/P-states disabled/Deep C-states disabled). Since C6Awarm has less power consumption than C1 (0.35W vs 1.44W), we expect C6Awarm to have more power savings than the baseline. Additionally, the performance impact due to the increase in the transition time of C6Awarm is only 90ns. The performance degradation caused by the 1% reduction in the clock frequency due to power gating depends on the sensitivity of each workload to the clock frequency.

Figure 10(a) and (b) show the average response time on the client side and the server side, respectively. For the server side, the performance degradation caused by the reduction on the

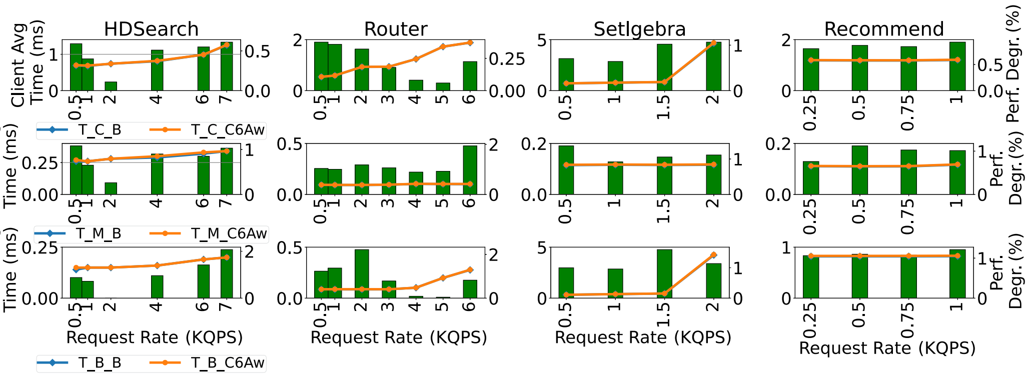

Fig. 11. Perf. degr. client/server side for MicroSuite when C6Awarm is enabled. T_C_*, Client time baseline/C6Awarm; T_M_*, Midtier time baseline/C6Awarm; T_B_*, Bucket time baseline/C6Awarm.

clock frequency ranges from 1.5% to 2%. As expected, the server-side reduction is higher than the client side, which is around 0.1%–0.2%. In the case of Memcached, the end-to-end response time is primarily dominated by the network latency. Specifically, 10% of the actual end-to-end time is spent on processing. As a result, the degradation experienced by the client is 1/10 of the degradation experienced by the server. The increase on the average response time observed in high loads is primarily a result of the increase in queueing overhead rather than the additional overhead of the new core C-state architecture for C6Awarm.

Figure 10(c) and (d) present the average core power savings and the core C-state residency, respectively. The power savings go from 72% to 15% while we increase the load of the system from 2% to 56%. For higher load, the opportunity to enter C6Awarm is less, as the activity of the system increases and as a result the idle time gets smaller. Even with a 50% average utilization, our AW architecture continues to have significant power savings (15%). Since the representative utilization of latency-critical applications is 5%–25% [7, 64, 65], our AW architecture provides significant power savings (72%–36%) for these applications.

It is important to note that C6Awarm evaluation of Memcached against a performance-oriented baseline provides more power savings than against a power-oriented baseline. The performance-oriented baseline has only core C-state C1 enabled, and as a result, all idle time is attributed to C1 offering significant power savings for C6Awarm (1.44W vs 0.35W compared to 0.1W vs 0.35W for a power-oriented baseline). For the medium-range utilization points, the power savings of the power- and performance-oriented baseline are close. This is because even though the core C-states are enabled for the power-optimized baseline, the system during medium utilization resides in C1/C1E and not in C6. The evaluation shows that AW compared to a system optimized for power consumption and a system optimized for performance provides up to 36% and 72% power savings, respectively.

We conclude that C6Awarm significantly reduces the core average power consumption of a Memcached service across various load levels with minimal performance overhead over the baseline configuration at which C1 is enabled, P-states are disabled, and Turbo is disabled.

6.3 Analysis of Additional Workload

In this section, we evaluate AW benefit for servers running applications with multi-tier microservice-based software architecture. Figure 11 presents the client and server average response time for all benchmarks of MicroSuite for C6Awarm compared to the baseline (Turbo disabled, C1 enabled, P-states disabled). The first row of the graph represents the client average

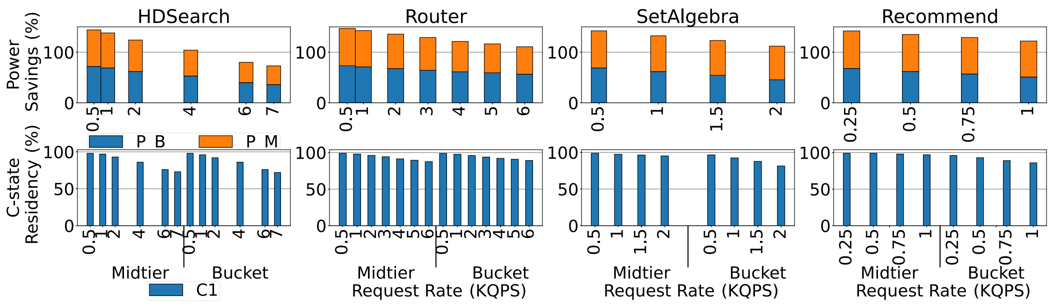

Fig. 12. Power savings midtier and bucket server for MicroSuite when C6Awarm is enabled. P_M, Midtier power savings; P_B: Bucket power savings.

response time, and the rest of the rows represent the midtier (managing both incoming and outgoing requests) and bucket (leaf-managing incoming requests) average response time, respectively. Each of the benchmarks exhibit different sensitivity to frequency degradation as the performance impact ranges from 0.2% to 2.2% for both midtier and bucket. As expected, the impact on the client side is less than the server side (0.1%–1.1%) since a significant portion of the end-to-end time is spend on the network communication. Contrary to Memcached, these benchmarks (except for Router) have higher client overhead (0.1%–0.6%/0.6%–1.1%/0.8%–0.92% for HDSearch/SetAlgebra/Recommend vs 0.05%–0.4% for Router). This is because a bigger percentage of the end-to-end response time is spent on computation (50%/60%/80% for HDSearch/SetAlgebra/Recommend vs 20%–30% for Router vs 10% for Memcached).

Figure 12 shows the core average power savings for each benchmark and tier (midtier, bucket) along with the core C-state residency. Like Memcached, the power savings go from 70% to 37% while we increase the load of the system from 2% to 27%. The increase in the load reduces the opportunity for the system to enter C1/C6Awarm, which reduces AW power savings.

We conclude that AW with disabled P-states, disabled Turbo, and C0/C6Awarm enabled significantly improves core average power for the MicroSuite benchmarks as compared to the baseline.

7 Related Work

As far as we know, this is the first work to introduce a core power state that provides both low transition/cold-start latency and low power consumption. While low server efficiency for latency-critical workloads has been studied before, previous work proposed management and scheduling techniques to mitigate the problem.

Modern Cloud Applications. Interactive latency-sensitive cloud applications are gradually shifting to a modular architecture based on loosely coupled microservices to meet their software maintenance, scalability, and availability requirements [62]. However, the decoupled nature of microservices exacerbates the strict tail latency requirements of such applications. Recent work [65] that characterizes microservices at Alibaba clusters shows that a user request relies on multiple microservices, whose individual times can accumulate into significant end-to-end latency. The same Alibaba study reveals that servers running latency-sensitive microservices operate at 10% utilization to keep the tail under control. This aligns with works from industry and academia that report the utilization of latency-sensitive applications to be 5%–20% [7, 64, 65]. Servers at low utilization have poor energy efficiency, partly because they disable deep C-states to avoid several microsecond penalty of transitioning out of such deep C-states [56]. This emphasizes the need for deep C-states with low transition latency and high power savings.

Fine-Grained, Latency-Aware DVFS Management. Besides C-states, the other major power management feature of modern processors is DVFS. Previous work proposes fine-grained DVFS control to save power while avoiding excessive latency degradation. Rubik [54] scales core frequency at the sub-millisecond scale based on a statistical performance model to save power while meeting a target tail latency. Swan [98] extends this idea to computational sprinting [77, 78] (e.g., Intel Turbo Boost [80, 81]): requests are initially served on a core operating at low frequency, and depending on the load, Swan scales up the frequency (including sprinting levels) to catch up and meet latency requirements. The new C6Awarm state of AW facilitates the effective use of idle states and could make a simple race-to-halt approach [32] more attractive than complex DVFS management techniques.

Workload-Aware Idle State Management. Various proposals exist for techniques that profile incoming request streams and use that information to improve power management. SleepScale [63] is a runtime power management technique that selects the most efficient C-state and DVFS setting for a given QoS constraint based on workload profiling information. CARB [97] packs requests into a subset of cores, while limiting latency degradation, so that the other cores have longer quiet times and can transition to deeper C-states. The idea of packing requests into a subset of active cores to extend idle periods on other cores is explored by works focusing on both C-states and DVFS management [5, 10, 11]. These proposals are orthogonal to AW: while C6Awarm can provide most of the benefits of a deep idle state at lower latency, advanced and workload-aware sleep management techniques can bring extra power savings by enabling cores and/or system to enter traditional deeper C-states for a longer time [3, 73]. Memory power management techniques [13, 14] have also been proposed to reduce system energy consumption and are complementary to our work.

Cold-Start Latency. The performance effects of the alteration of the microarchitectural state between active periods of an application has been studied either through core migration [12, 83] or power gating [2, 4, 10]. In Lukewarm, Schall et al. [83] noticed that the microarchitectural state of the core changes between invocations of the same function increasing the CPI. They identified the L1I cache as the main contributor and proposed an instruction prefetcher to speed up the warm-up time. Another study [12] investigated various factors (i.e., migration frequency) impacting the warm-up time of a recently migrated thread/process. According to their findings, for the warm-up time to reduce, the core must maintain the predictor state of the recently migrated thread/process, caches should be coherent between active and inactive cores, and migration should occur at most every 160K cycles. AW agrees with prior works indicating that cold-start latency affects performance in some scenarios. Additionally, it introduces a solution integrated into the C-state architecture that reduces cold-start latency with minimal impact on power, performance, and area.

8 Conclusion

AW is a novel architecture that reduces the transition and the cold-start latency to/from deep power states while maintaining the power savings of deep sleep states. Our evaluation showed that depending on the enabled core C-states (C6Awarm/C6AwarmE), and the baseline configuration, AW can reduce up to 70% the power consumption of a core with 2% end-to-end performance degradation. The results are encouraging toward the adoption of AW in datacenter servers running latency-critical applications based on microservices.

Acknowledgments

We would like to thank the anonymous reviewers for their thorough and insightful comments on earlier versions of this article.

References

[1] Charbel J. Akl, Rafic A. Ayoubi, and Magdy A. Bayoumi. 2009. An effective staggered-phase damping technique for suppressing power-gating resonance noise during mode transition. In *Proceedings of the 10th International Symposium on Quality Electronic Design (ISQED'09)*. IEEE, 116–119. https://doi.org/10.1109/ISQED.2009.4810280

[2] Hrishikesh Amur, Ripal Nathuji, Mrinmoy Ghosh, Karsten Schwan, and Hsien-Hsin Lee. 2008. IdlePower: Application-aware management of processor idle states. In *Proceedings of the Workshop on Managed Many-Core Systems (MMCS'08)*.

[3] Georgia Antoniou, Haris Volos, Davide B. Bartolini, Tom Rollet, Yiannakis Sazeides, and Jawad Haj Yahya. 2022. AgilePkgC: An agile system idle state architecture for energy proportional datacenter servers. In *Proceedings of the 2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO'22)*. IEEE, 851–867. https://doi.org/10.1109/MICRO56248.2022.00065

[4] Manish Arora, Srilatha Manne, Indrani Paul, Nuwan Jayasena, and Dean M. Tullsen. 2015. Understanding idle behavior and power gating mechanisms in the context of modern benchmarks on CPU-GPU integrated systems. In *Proceedings of the 2015 IEEE International Symposium on High Performance Computer Architecture (HPCA'15)*. IEEE, 366–377. https://doi.org/10.1109/HPCA.2015.7056047

[5] Esmail Asyabi, Azer Bestavros, Erfan Sharafzadeh, and Timothy Zhu. 2020. Peafowl: In-application CPU scheduling to reduce power consumption of in-memory key-value stores. In *Proceedings of the 11th ACM Symposium on Cloud Computing (SoCC'20)*. ACM, New York, NY, USA.

[6] Berk Atikoglu, Yuehai Xu, Eitan Frachtenberg, Song Jiang, and Mike Paleczny. 2012. Workload analysis of a large-scale key-value store. *ACM SIGMETRICS Performance Evaluation Review* 40, 1 (2012), 53–64.

[7] Luiz André Barroso, Jeffrey Dean, and Urs Hölzle. 2003. Web search for a planet: The Google cluster architecture. *IEEE Micro* 23, 2 (March 2003), 22–28. https://doi.org/10.1109/MM.2003.1196112

[8] Rakesh Chadha and J. Bhasker. 2013. Architectural techniques for low power. In *An ASIC Low Power Primer*. Springer, Berlin, Germany.

[9] Wei Chen, Szu-Liang Chen, Siufu Chiu, Raghuraman Ganesan, Venkata Lukka, Wei Wing Mar, and Stefan Rusu. 2013. A 22nm 2.5 MB slice on-die L3 cache for the next generation Xeon® processor. In *Proceedings of the Symposium on VLSI Circuits*. IEEE.

[10] Chih-Hsun Chou, Laxmi N. Bhuyan, and Daniel Wong. 2019. μDPM: Dynamic power management for the microsecond era. In *Proceedings of the 2019 IEEE International Symposium on High Performance Computer Architecture (HPCA'19)*. IEEE.

[11] Chih-Hsun Chou, Daniel Wong, and Laxmi N. Bhuyan. 2016. DynSleep: Fine-grained power management for a latency-critical data center application. In *Proceedings of the 2016 International Symposium on Low Power Electronics and Design (ISLPED'16)*. ACM, New York, NY, USA.

[12] Theofanis Constantinou, Yiannakis Sazeides, Pierre Michaud, Damien Fetis, and Andre Seznec. 2005. Performance implications of single thread migration on a chip multi-core. *ACM SIGARCH Computer Architecture News* 33, 4 (Nov. 2005), 80–91. https://doi.org/10.1145/1105734.1105745

[13] Howard David, Chris Fallin, Eugene Gorbatov, Ulf R. Hanebutte, and Onur Mutlu. 2011. Memory power management via dynamic voltage/frequency scaling. In *Proceedings of the 8th ACM International Conference on Autonomic Computing (ICAC'11)*. ACM, New York, NY, USA.

[14] Qingyuan Deng, David Meisner, Luiz Ramos, Thomas F. Wenisch, and Ricardo Bianchini. 2011. MemScale: Active low-power modes for main memory. *ACM SIGARCH Computer Architecture News* 39, 1 (March 2011), 225–238. https://doi.org/10.1145/1961295.1950392

[15] Lide Duan, Dongyuan Zhan, and Justin Hohnerlein. 2015. Optimizing cloud data center energy efficiency via dynamic prediction of CPU idle intervals. In *Proceedings of the 2015 IEEE 8th International Conference on Cloud Computing*. IEEE, 985–988. https://doi.org/10.1109/CLOUD.2015.133

[16] Dmitry Duplyakin, Robert Ricci, Aleksander Maricq, Gary Wong, Jonathon Duerig, Eric Eide, Leigh Stoller, Mike Hibler, David Johnson, Kirk Webb, Aditya Akella, Kuangching Wang, Glenn Ricart, Larry Landweber, Chip Elliott, Michael Zink, Emmanuel Cecchet, Snigdhaswin Kar, and Prabodh Mishra. 2019. The design and operation of CloudLab. In *Proceedings of the USENIX Annual Technical Conference (ATC'19)*. 1–14. https://www.flux.utah.edu/paper/duplyakin-atc19

[17] Ali M. El-Husseini and Matthew Morrise. 2011. Clocking design automation in Intel's Core i7 and future designs. In *Proceedings of the 2011 IEEE/ACM International Conference on Computer-Aided Design (ICCAD'11)*. IEEE.

[18] Mark Ermolov, Dmitry Sklyarov, and Maxim Goryachy. 2023. Undocumented x86 instructions to control the CPU at the microarchitecture level in modern Intel processors. *Journal of Computer Virology and Hacking Techniques* 19, 3 (Sept. 2023), 351–365. https://doi.org/10.1007/s11416-022-00438-x

[19] Eyal Fayneh, Marcelo Yuffe, Ernest Knoll, Michael Zelikson, Muhammad Abozaed, Yair Talker, Ziv Shmuely, and Saher Abu Rahme. 2016. 4.1 14nm 6th-generation Core processor SoC with low power consumption and improved performance. In *Proceedings of the 2016 IEEE International Solid-State Circuits Conference (ISSCC'16)*. IEEE.

[20] Maria Fazio, Antonio Celesti, Rajiv Ranjan, Chang Liu, Lydia Chen, and Massimo Villari. 2016. Open issues in scheduling microservices in the cloud. *IEEE Cloud Computing* 3, 5 (2016), 81–88. https://doi.org/10.1109/MCC.2016.112

[21] Brad Fitzpatrick. 2023. Memcached: A distributed memory object caching system. Retrieved July 4, 2024 from https://memcached.org/

[22] K. Flautner, Nam Sung Kim, S. Martin, D. Blaauw, and T. Mudge. 2002. Drowsy caches: Simple techniques for reducing leakage power. In *Proceedings 29th Annual International Symposium on Computer Architecture*. IEEE, 148–157. https://doi.org/10.1109/ISCA.2002.1003572

[23] David Flynn, Rob Aitken, Alan Gibbons, and Kaijian Shi. 2007. *Low Power Methodology Manual: for System-on-Chip Design*. Springer Science & Business Media, Berlin, Germany.

[24] Alex Gendler, Ernest Knoll, and Yiannakis Sazeides. 2021. I-DVFS: Instantaneous frequency switch during dynamic voltage and frequency scaling. *IEEE Micro* 41, 5 (2021), 76–84. https://doi.org/10.1109/MM.2021.3096655

[25] Gianfranco Gerosa, Steve Curtis, Mike D'Addeo, Bo Jiang, Belliappa Kuttanna, Feroze Merchant, Binta Patel, Mohammed Taufique, and Haytham Samarchi. 2008. A sub 2W low power IA processor for mobile internet devices in 45nm Hi-K metal gate CMOS. In *Proceedings of the 2008 IEEE Asian Solid-State Circuits Conference*. IEEE, 17–20. https://doi.org/10.1109/ASSCC.2008.4708718

[26] Corey Gough, Ian Steiner, and Winston Saunders. 2015. CPU power management. In *Energy Efficient Servers: Blueprints for Data Center Optimization*. Springer, Berlin, Germany.

[27] Ed Grochowski, David Ayers, and Vivek Tiwari. 2002. Microarchitectural simulation and control of di/dt-induced power supply voltage variation. In *Proceedings of the 2002 IEEE International Symposium on High Performance Computer Architecture (HPCA'02)*. IEEE.

[28] Meeta S. Gupta, Krishna K. Rangan, Michael D. Smith, Gu-Yeon Wei, and David Brooks. 2008. DeCoR: A delayed commit and rollback mechanism for handling inductive noise in processors. In *Proceedings of the 2008 IEEE International Symposium on High Performance Computer Architecture (HPCA'08)*. IEEE.

[29] Linley Gwennap. 1997. *P6 Microcode Can Be Patched. Microprocessor Report*. Microdesign Resources.

[30] Daniel Hackenberg, Daniel Molka, and Wolfgang E. Nagel. 2009. Comparing cache architectures and coherency protocols on x86-64 multicore SMP systems. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'09)*. IEEE.

[31] J. Haj-Yahya, M. Alser, J. S. Kim, L. Orosa, E. Rotem, A. Mendelson, A. Chattopadhyay, and O. Mutlu. 2020. FlexWatts: A power- and workload-aware hybrid power delivery network for energy-efficient microprocessors. In *Proceedings of the 2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'20)*. IEEE, 1051–1066. https://doi.org/10.1109/MICRO50266.2020.00088

[32] Jawad Haj-Yahya, Avi Mendelson, Yosi Ben Asher, and Anupam Chattopadhyay. 2018. Power management of modern processors. In *Energy Efficient High Performance Processors*. Computer Architecture and Design Methodologies. Springer, 1–55.

[33] J. Haj-Yahya, L. Orosa, J. S. Kim, J. Gomez Luna, A. Yaglikci, M. Alser, I. Puddu, and O. Mutlu. 2021. IChannels: Exploiting current management mechanisms to create covert channels in modern processors. In *Proceedings of the 2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA'21)*. IEEE, 985–998. https://doi.org/10.1109/ISCA52012.2021.00081

[34] Jawad Haj-Yahya, Efraim Rotem, Avi Mendelson, and Anupam Chattopadhyay. 2019. A comprehensive evaluation of power delivery schemes for modern microprocessors. In *Proceedings of the 20th International Symposium on Quality Electronic Design (ISQED'19)*. IEEE.

[35] Jawad Haj-Yahya, Yanos Sazeides, Mohammed Alser, Efraim Rotem, and Onur Mutlu. 2020. Techniques for reducing the connected-standby energy consumption of mobile devices. In *Proceedings of the 2020 IEEE International Symposium on High Performance Computer Architecture (HPCA'20)*. IEEE, Washington, DC, USA.

[36] Jawad Haj-Yihia, Ahmad Yasin, Yosi Ben Asher, and Avi Mendelson. 2016. Fine-grain power breakdown of modern out-of-order cores and its implications on Skylake-based systems. *ACM Transactions on Architecture and Code Optimization* 13, 4 (Dec. 2016), Article 56, 25 pages. https://doi.org/10.1145/3018112

[37] Jawad Haj-Yihia, Ahmad Yasin, and Yosi Ben-Asher. 2015. DOEE: Dynamic optimization framework for better energy efficiency. In *Proceedings of the Symposium on High Performance Computing(HPC'15)*. 107–114.

[38] Brett Howse and Ryan Smith. 2015. Tick Tock on the Rocks: Intel Delays 10nm, Adds 3rd Gen 14nm Core Product "Kaby Lake." Retrieved July 4, 2024 from https://www.anandtech.com/show/9447/intel-10nm-and-kaby-lake

[39] Mo Huang, Yan Lu, Sai-Weng Sin, U. Seng-Pan, and Rui P. Martins. 2016. A fully integrated digital LDO with coarse-fine-tuning and burst-mode operation. *IEEE Transactions on Circuits and Systems II: Express Briefs* 63, 7 (2016), 683–687. https://doi.org/10.1109/TCSII.2016.2530094

[40] Min Huang, Moty Mehalel, Ramesh Arvapalli, and Songnian He. 2013. An energy efficient 32-nm 20-mb shared on-die L3 cache for Intel® Xeon® processor E5 family. *IEEE Journal of Solid-State Circuits* 48, 8 (2013), 1954–1962. https://doi.org/10.1109/JSSC.2013.2258815

[41] Intel. 2019. Intel 64 and IA-32 Architectures Software Developer's Manual Volume 3A, 3B, and 3C. Retrieved July 4, 2024 from https://intel.ly/3gVj2Fy

[42] Intel. 2020. 6th Generation Intel® Processor for U/Y-Platforms Datasheet. Retrieved July 4, 2024 from https://intel.ly/37rtnU7

[43] Intel. 2021. Intel Xeon Silver 4114 Processor. Retrieved July 4, 2024 from https://intel.ly/3x7rx7N

[44] Intel. 2021. *Intel® Xeon® Processor E7-8800/4800/2800 v2 Product Family*. Datasheet–Vol. 1. Intel. https://intel.ly/2ZGA9FJ

[45] Intel Corporation. n.d. Intel Idle Driver for Linux. Retrieved July 4, 2024 from https://bit.ly/3GKRJbK

[46] Sanjeev Jahagirdar, Vargbese George, John B. Conrad, Robert Milstrey, Stephen A. Fischer, Alon Naveh, and Shai Rotem. 2012. Method and apparatus for a zero voltage processor sleep state. US Patent App. 13/220,413.

[47] Sanjeev Jahagirdar, Varghese George, Inder Sodhi, and Ryan Wells. 2012. Power management of the third generation Intel Core micro architecture formerly codenamed Ivy Bridge. In *Proceedings of the 2012 IEEE Hot Chips 24 Symposium (HCS'12)*. IEEE.

[48] Kwangok Jeong, Andrew B. Kahng, Seokhyeong Kang, Tajana S. Rosing, and Richard Strong. 2012. MAPG: Memory access power gating. In *Proceedings of the 2012 Design, Automation, and Test in Europe Conference and Exhibition (DATE'12)*. IEEE.

[49] Hailin Jiang, Malgorzata Marek-Sadowska, and Sani R. Nassif. 2005. Benefits and costs of power-gating technique. In *Proceedings of the 2005 International Conference on Computer Design (ICCD'05)*. IEEE, 559–566. https://doi.org/10.1109/ICCD.2005.34

[50] Chaoqiang Jin, Xuelian Bai, Chao Yang, Wangxin Mao, and Xin Xu. 2020. A review of power consumption models of servers in data centers. *Applied Energy* 265 (2020), 114806. https://doi.org/10.1016/j.apenergy.2020.114806

[51] Jithin Jose, Hari Subramoni, Miao Luo, Minjia Zhang, Jian Huang, Md. Wasi-Ur Rahman, Nusrat S. Islam, Xiangyong Ouyang, Hao Wang, Sayantan Sur, and K. Dhabaleswar Panda. 2011. Memcached design on high performance RDMA capable interconnects. In *Proceedings of the 2011 International Conference on Parallel Processing (ICPP'11)*. IEEE.

[52] Ravi Jotwani, Sriram Sundaram, Stephen Kosonocky, Alex Schaefer, Victor Andrade, Greg Constant, Amy Novak, and Samuel Naffziger. 2010. An x86-64 core implemented in 32nm SOI CMOS. In *Proceedings of the 2010 IEEE International Solid-State Circuits Conference (ISSCC'10)*. IEEE.

[53] Andrew B. Kahng, Seokhyeong Kang, Tajana Rosing, and Richard Strong. 2012. TAP: Token-based adaptive power gating. In *Proceedings of the 2012 International Symposium on Low Power Electronics and Design (ISLPED'12)*. ACM, New York, NY, USA.

[54] Harshad Kasture, Davide B. Bartolini, Nathan Beckmann, and Daniel Sanchez. 2015. Rubik: Fast analytical power management for latency-critical systems. In *Proceedings of the 2015 48th IEEE/ACM International Symposium on Microarchitecture (MICRO'15)*. IEEE.

[55] Keysight. 2019. Diagram for Skylake-SP Core. Retrieved July 4, 2024 from https://intel.ly/3Ctjwfr

[56] Keysight. n.d. BIOS Performance and Power Tuning Guidelines for Dell PowerEdge 12th Generation Servers. Retrieved November 30, 2021 from https://bit.ly/3llqoFh

[57] Philipp Koppe, Benjamin Kollenda, Marc Fyrbiak, Christian Kison, Robert Gawlik, Christof Paar, and Thorsten Holz. 2017. Reverse engineering x86 processor microcode. In *Proceedings of the 26th USENIX Security Symposium(USENIX Security'17)*. 1163–1180.

[58] David E. Lackey, Paul S. Zuchowski, Thomas R. Bednar, Douglas W. Stout, Scott W. Gould, and John M. Cohn. 2002. Managing power and performance for system-on-chip designs using voltage islands. In *Proceedings of the 2002 IEEE/ACM International Conference on Computer-Aided Design (ICCAD'02)*. IEEE.

[59] George Lakkas. 2016. MOSFET power losses and how they affect power-supply efficiency. *Analog Applications Journal* 1Q 2016 (2016), 22–26.

[60] Lenovo. n.d. Tuning UEFI Settings for Performance and Energy Efficiency on Intel Xeon Scalable Processor-Based ThinkSystem Servers. Retrieved November 30, 2021 from https://lenovopress.com/lp1477.pdf

[61] Jacob Leverich. 2014. Mutilate: High-Performance Memcached Load Generator. Retrieved July 4, 2024 from https://github.com/leverich/mutilate

[62] James Lewis and Martin Fowler. 2023. Microservices. Retrieved July 4, 2024 from https://martinfowler.com/articles/microservices.html

[63] Yanpei Liu, Stark C. Draper, and Nam Sung Kim. 2014. SleepScale: Runtime joint speed scaling and sleep states management for power efficient data centers. *ACM SIGARCH Computer Architecture News* 42, 3 (2014), 313–324.

[64] David Lo, Liqun Cheng, Rama Govindaraju, Luiz André Barroso, and Christos Kozyrakis. 2014. Towards energy proportionality for large-scale latency-critical workloads. In *Proceedings of the 2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA'14)*. IEEE.

[65] Shutian Luo, Huanle Xu, Chengzhi Lu, Kejiang Ye, Guoyao Xu, Liping Zhang, Yu Ding, Jian He, and Chengzhong Xu. 2021. Characterizing microservice dependency and performance: Alibaba trace analysis. In *Proceedings of the 12th ACM Symposium on Cloud Computing(SoCC'21)*. ACM, New York, NY, USA, 412–426. https://doi.org/10.1145/3472883.3487003

[66] Kosta Luria, Joseph Shor, Michael Zelikson, and Alex Lyakhov. 2016. Dual-mode low-drop-out regulator/power gate with linear and on-off conduction for microprocessor core on-die supply voltages in 14 nm. *IEEE Journal of Solid-State Circuits* 51, 3 (2016), 752–762. https://doi.org/10.1109/JSSC.2015.2512387

[67] Hamid Mahmoodi-Meimand and Kaushik Roy. 2004. Data-retention flip-flops for power-down applications. In *Proceedings of the 2004 IEEE International Symposium on Circuits and Systems (ISCAS'04)*. IEEE.

[68] Julius Mandelblat. 2015. Technology Insight: Intel's Next Generation Microarchitecture Code Name Skylake. Retrieved July 4, 2024 from https://en.wikichip.org/w/images/8/8f/Technolog_Insight_Intel%E2%80%99s_Next_Generation_Microarchitecture_Code_Name_Skylake.pdf

[69] Amirhossein Mirhosseini, Akshitha Sriraman, and Thomas F. Wenisch. 2019. Enhancing server efficiency in the face of killer microseconds. In *Proceedings of the 2019 IEEE International Symposium on High Performance Computer Architecture (HPCA'19)*. IEEE.

[70] Daniel Molka, Daniel Hackenberg, Robert Schone, and Matthias S. Muller. 2009. Memory performance and cache coherency effects on an Intel Nehalem multiprocessor system. In *Proceedings of the 2009 18th International Conference on Parallel Architectures and Compilation Techniques (PACT'09)*. IEEE.

[71] Ankireddy Nalamalpu, Nasser Kurd, Anant Deval, Chris Mozak, Jonathan Douglas, Ashish Khanna, Fabrice Paillet, Gerhard Schrom, and Boyd Phelps. 2015. Broadwell: A family of IA 14nm processors. In *Proceedings of the Symposium on VLSI Circuits*. IEEE.

[72] Dmitry Namiot and Manfred Sneps-Sneppe. 2014. On micro-services architecture. *International Journal of Open Information Technologies* 2 (2014), 24–27.

[73] Steven Pelley, David Meisner, Thomas F. Wenisch, and James W. VanGilder. 2009. Understanding and abstracting total data center power. In *Proceedings of the Workshop on Energy-Efficient Design (WEED'09)*.

[74] Dustin Peterson and Oliver Bringmann. 2019. Fully-automated synthesis of power management controllers from UPF. In *Proceedings of the 24th Asia and South Pacific Design Automation Conference (ASP-DAC'19)*. ACM, New York, NY, USA.

[75] Joseph Rabinowicz and Shlomo Greenberg. 2021. A new physical design flow for a selective state retention based approach. *Journal of Low Power Electronics and Applications* 11, 3 (2021), 35. https://doi.org/10.3390/jlpea11030035

[76] Kaladhar Radhakrishnan, Madhavan Swaminathan, and Bidyut K. Bhattacharyya. 2021. Power delivery for high-performance microprocessors—Challenges, solutions, and future trends. *IEEE Transactions on Components, Packaging and Manufacturing Technology* 11, 4 (2021), 655–671. https://doi.org/10.1109/TCPMT.2021.3065690

[77] Arun Raghavan, Yixin Luo, Anuj Chandawalla, Marios Papaefthymiou, Kevin P. Pipe, Thomas F. Wenisch, and Milo M. K. Martin. 2013. Designing for responsiveness with computational sprinting. *IEEE Micro* 33, 3 (2013), 8–15. https://doi.org/10.1109/MM.2013.51

[78] Arun Raghavan, Yixin Luo, Anuj Chandawalla, Marios Papaefthymiou, Kevin P. Pipe, Thomas F. Wenisch, and Milo M. K. Martin. 2012. Computational sprinting. In *Proceedings of the 2012 IEEE International Symposium on High Performance Computer Architecture (HPCA'12)*. IEEE, 1–12. https://doi.org/10.1109/HPCA.2012.6169031

[79] Aaron Rogers, David Kaplan, Eric Quinnell, and Bill Kwan. 2012. The core-C6 (CC6) sleep state of the AMD Bobcat x86 microprocessor. In *Proceedings of the 2012 International Symposium on Low Power Electronics and Design (ISLPED'12)*. ACM, New York, NY, USA.

[80] Efraim Rotem, Alon Naveh, Avinash Ananthakrishnan, Eliezer Weissmann, and Doron Rajwan. 2012. Power-management architecture of the Intel microarchitecture code-named Sandy Bridge. *IEEE Micro* 32, 2 (2012), 20–27.

[81] Efi Rotem, Alon Naveh, Doron Rajwan, Avinash Ananthakrishnan, and Eli Weissmann. 2011. Power management architecture of the 2nd generation Intel® Core microarchitecture, formerly codenamed Sandy Bridge. In *Proceedings of the 2011 IEEE Hot Chips 23 Symposium (HCS'11)*. IEEE, 1–33. https://doi.org/10.1109/HOTCHIPS.2011.7477510

[82] Stefan Rusu, Harry Muljono, David Ayers, Simon Tam, Wei Chen, Aaron Martin, Shenggao Li, Sujal Vora, Raj Varada, and Eddie Wang. 2014. 5.4 Ivytown: A 22nm 15-core enterprise Xeon® processor family. In *Proceedings of the 2014 IEEE International Solid-State Circuits Conference (ISSCC'14)*. IEEE.

[83] David Schall, Artemiy Margaritov, Dmitrii Ustiugov, Andreas Sandberg, and Boris Grot. 2022. Lukewarm serverless functions: Characterization and optimization. In *Proceedings of the 49th Annual International Symposium on Computer Architecture(ISCA'22)*. ACM, New York, NY, USA, 757–770. https://doi.org/10.1145/3470496.3527390

[84] Robert Schöne, Thomas Ilsche, Mario Bielert, Andreas Gocht, and Daniel Hackenberg. 2019. Energy efficiency features of the Intel Skylake-SP processor and their impact on performance. In *Proceedings of the 2019 International Conference on High Performance Computing and Simulation (HPCS'19)*. IEEE, 399–406.

[85] Robert Schöne, Daniel Molka, and Michael Werner. 2015. Wake-up latencies for processor idle states on current x86 processors. *Computer Science—Research and Development* 30, 2 (2015), 219–227.

[86] Erfan Sharafzadeh, Seyed Alireza Sanaee Kohroudi, Esmail Asyabi, and Mohsen Sharifi. 2019. Yawn: A CPU idle-state governor for datacenter applications. In *Proceedings of the 10th ACM SIGOPS Asia-Pacific Workshop on Systems(APSys'19)*. ACM, New York, NY, USA, 91–98. https://doi.org/10.1145/3343737.3343740

[87] Akshitha Sriraman and Thomas F. Wenisch. 2018. μ Suite: A benchmark suite for microservices. In *Proceedings of the 2018 IEEE International Symposium on Workload Characterization (IISWC'18)*. IEEE, 1–12. https://doi.org/10.1109/IISWC.2018.8573515

[88] Simon M. Tam, Harry Muljono, Min Huang, Sitaraman Iyer, Kalapi Royneogi, Nagmohan Satti, Rizwan Qureshi, Wei Chen, Tom Wang, Hubert Hsieh, Sujal Vora, and Eddie Wang. 2018. SkyLake-SP: A 14nm 28-core Xeon® processor. In *Proceedings of the 2018 IEEE International Solid-State Circuits Conference (ISSCC'18)*. IEEE, 34–36. https://doi.org/10.1109/ISSCC.2018.8310170

[89] Thiago Lara Vasques, Pedro Moura, and Aníbal de Almeida. 2019. A review on energy efficiency and demand response with focus on small and medium data centers. *Energy Efficiency* 12, 5 (June 2019), 1399–1428. https://doi.org/10.1007/s12053-018-9753-2

[90] WikiChip. 2021. Mesh Interconnect Architecture—Intel. Retrieved November 30, 2021 from https://en.wikichip.org/wiki/intel/mesh_interconnect_architecture

[91] WikiChip. 2021. Skylake (Server)–Microarchitectures–Intel. Retrieved July 4, 2024 from https://bit.ly/2MHEWkj

[92] Jawad Haj Yahya, Haris Volos, Davide B. Bartolini, Georgia Antoniou, Jeremie S. Kim, Zhe Wang, Kleovoulos Kalaitzidis, Tom Rollet, Zhirui Chen, Ye Geng, Onur Mutlu, and Yiannakis Sazeides. 2022. AgileWatts: An energy-efficient CPU core idle-state architecture for latency-sensitive server applications. In *Proceedings of the 2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO'22)*. IEEE, 835–850. https://doi.org/10.1109/MICRO56248.2022.00063

[93] Juncheng Yang, Yao Yue, and K. V. Rashmi. 2020. A large scale analysis of hundreds of in-memory cache clusters at Twitter. In *Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI'20)*. 191–208.

[94] Ahmad Yasin. 2014. A top-down method for performance analysis and counters architecture. In *Proceedings of the 2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'94)*. IEEE, 35–44.

[95] Ahmad Yasin, Jawad Haj-Yahya, Yosi Ben-Asher, and Avi Mendelson. 2019. A metric-guided method for discovering impactful features and architectural insights for Skylake-based processors. *ACM Transactions on Architecture and Code Optimization* 16, 4 (2019), Article 46, 25 pages.

[96] Ahmad Yasin, Nir Rosenzweig, Eliezer Weissmann, and Efraim Rotem. 2017. Performance scalability prediction. US Patent 9,829,957.

[97] Xin Zhan, Reza Azimi, Svilen Kanev, David Brooks, and Sherief Reda. 2017. CARB: A C-state power management arbiter for latency-critical workloads. *IEEE Computer Architecture Letters* 16, 1 (2017), 6–9. https://doi.org/10.1109/LCA.2016.2537802

[98] Liang Zhou, Laxmi N. Bhuyan, and K. K. Ramakrishnan. 2020. Swan: A two-step power management for distributed search engines. In *Proceedings of the 2020 International Symposium on Low Power Electronics and Design (ISLPED'20)*. ACM, New York, NY, USA.

[99] Brian Zimmer, Pi-Feng Chiu, Borivoje Nikolić, and Krste Asanović. 2017. Reprogrammable redundancy for SRAM-based cache Vmin reduction in a 28-nm RISC-V processor. *IEEE Journal of Solid-State Circuits* 52, 10 (2017), 121–124.

Received 20 October 2023; revised 14 April 2024; accepted 17 May 2024