AgilePkgC: An Agile System Idle State Architecture for Energy Proportional Datacenter Servers

Georgia Antoniou†‡, Haris Volos‡, Davide B. Bartolini§
Tom Rollet§, Yiannakis Sazeides§, Jawad Haj Yahya§

$§$Huawei Technologies - Zurich Research Center
‡University of Cyprus

ABSTRACT

Modern user-facing applications deployed in datacenters use a distributed system architecture that exacerbates the latency requirements of their constituent microservices (30–250µs). Existing CPU power-saving techniques degrade the performance of these applications due to the long transition latency (order of 100µs) to wake up from a deep CPU idle state (C-state). For this reason, server vendors recommend only enabling shallow core C-states (e.g., CC1) for idle CPU cores, thus preventing the system from entering deep package C-states (e.g., PC6) when all CPU cores are idle. This choice, however, impairs server energy proportionality since power-hungry resources (e.g., IOs, uncore, DRAM) remain active even when there is no active core to use them. As we show, it is common for all cores to be idle due to the low average utilization (e.g., 5–20%) of datacenter servers running user-facing applications.

We propose to reap this opportunity with AgilePkgC (APC), a new package C-state architecture that improves the energy proportionality of server processors running latency-critical applications. APC implements PC1A (package C1 agile), a new deep package C-state that a system can enter once all cores are in a shallow C-state (i.e., CC1) and has a nanosecond-scale transition latency. APC is based on four key techniques. First, a hardware-based agile power management unit (APMU) rapidly detects when all cores enter a shallow core C-state (CC1) and trigger the system-level power savings control flow. Second, an IO Standby Mode (IOSM) that places IO interfaces (e.g., PCIe, DMI, UPI, DRAM) in shallow (nanosecond-scale transition latency) low-power modes. Third, a CLM Retention (CLMR) rapidly reduces the CLM (Cache-and-home-agent, Last-level-cache, and Mesh network-on-chip) domain’s voltage to its retention level, drastically reducing its power consumption. Fourth, APC keeps all system PLLs active in PC1A to allow nanosecond-scale exit latency by avoiding PLLs’ re-locking overhead.

Combining these techniques enables significant power savings while requiring less than 200ns transition latency, >250× faster than existing deep package C-states (e.g., PC6), making PC1A practical for datacenter servers. Our evaluation using Intel Skylake-based server shows that APC reduces the energy consumption of Memcached by up to 41% (25% on average) with <0.1% performance degradation. APC provides similar benefits for other representative workloads.

1. INTRODUCTION

The development of cloud applications running in datacenters is increasingly moving away from a monolithic to microservice software architecture to facilitate productivity [21, 48]. This comes at the expense of application performance becoming more vulnerable to events that result in "killer" microsecond scale idleness [9]. This is acute for user-facing applications with tight tail-latency requirements whereby serving a user query typically consists of executing numerous interacting microservices that explicitly communicate with each other [9, 10, 73]. The communication latency limits the time available to execute a microservice and magnifies the impact of microsecond scale idleness (e.g., events related to NVM, main memory access, and power management) [9, 15, 17]. This is further compounded by the dynamics of user-facing applications’ unpredictable and bursty load [16, 17, 65]. As a result, each microservice needs to operate under a tight (i.e., tens to hundreds of µs) latency requirement [17, 89].

One widely used method to ensure that microservices, and hence overall applications, meet their performance target is to execute them on servers that have low average utilization (5–20%) [47, 62, 91–94], leading to a busy/idle execution pattern [16, 17, 65] where cores are frequently idle. Ideally, each core should enter a low-power core C-state whenever it is idle, and the entire system should transition to a low-power package C-state whenever all cores are idle. However, the situation in modern datacenters is quite different.

Table 1: Power across existing package C-states and our new PC1A for our baseline server (details in Sec. 6).

| Package / cores C-state | Latency | SoC + DRAM power |
|-------------------------|---------|-----------------|
| PC0 / ≥1 CC0            | 0ns     | ≤85W + 7W        |
| PC0/die                 | 0ns     | 44W + 5.5W       |
| PC6                     | >50µs   | 12W + 0.5W       |
| PC1A / 10 CC1           | < 200ns | 27.5W + 1.6W     |

1We report worst-case entry+exit latency for the package C-state to open the path to memory; the overall latency to resume execution may be higher and include core C-state latency [26, 28].
2We refer to core C-states as CCx and package C-states as PCx; higher values of x indicate deeper, lower-power C-states.
high transition latency imposed by CC6 (and, subsequently, PC6), coupled with short and unpredictable request arrivals, severely reduces the usefulness of these deep C-states in datacenter servers. Server vendors recommend disabling deep core C-states in datacenters to prevent response-time degradation [46, 53, 54, 57]. Consequently, existing package C-states can never be entered even when all cores are idle in CC1 (e.g., Intel modern servers can only enter PC6 if all cores are in CC6) [26, 39]. This scenario in datacenter servers results in significant power waste as the uncore and other shared components (e.g., DRAM) fail to enter any low-power state when all cores are idle.

A seminal work by Google that discusses latency-critical applications states [62]: “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 typically 5%–20% to meet target tail latency requirements, as reported by multiple works from industry and academia [62, 91–94]. For example, recently, Alibaba reported that the utilization of servers running latency-critical applications is typically 10% [94]. Therefore, to improve the energy proportionality of servers running latency-critical microservice-based applications, it is crucial to address the more inefficient servers’ operating points, namely the low utilization, which is the focus of our study.

Prior work (reviewed in Sec. 8) proposes various management techniques to mitigate the inability of datacenter processors to leverage deep C-states effectively. In contrast, our goal is to directly address the root cause of the inefficiency, namely the high transition latency (tens of μs; see Table 1) of deep package C-states. To this end, we propose AgilePkgC (APC): a new package C-state architecture to improve the energy proportionality of server processors running latency-critical applications. APC introduces PC1A: a low-power package C-state with nanosecond-scale transition latency that the system can enter as soon as all cores enter shallow C-states (e.g., CC1, rather than after all cores enter deeper C-states, e.g., CC6, which are unreachable as they are normally disabled in server systems). A low-latency package C-state is crucial since periods of whole-system idleness (i.e., all cores idle) are even shorter and more unpredictable than idle periods of individual cores.

APC leverages four key power management techniques that differentiate PC1A from existing package C-states. 1) A hardware-based agile power management unit (APMU) to rapidly detect when all cores enter a shallow core C-state (CC1) and trigger a system-level power savings flow. 2) An IO Standby Mode (IOM) that places IO interfaces (e.g., PCIe, DMI, UPI, DRAM) in shallow (nanosecond-scale transition latency) low-power modes. 3) A CLM Retention (CLMR) that leverages the fast integrated voltage regulator [12, 67] to rapidly reduce the CLM (Cache-and-home-agent, Last-level-cache, and Mesh network-on-chip) domain’s voltage to its retention level, drastically reducing CLM’s power consumption. 4) APC keeps all system PLLs active in PC1A to allow nanosecond-scale exit latency by avoiding PLLs’ re-locking latency (a few microseconds). This approach significantly reduces transition latency at a minimal power cost, thanks to modern all-digital PLLs’ energy efficiency [25].

Our evaluation using Intel Skylake-based server shows that APC reduces the energy consumption of Memcached [2] by up to 41% (25% on average) with <0.1% performance degradation. APC provides similar benefits for other representative workloads. APC’s new package C-states, PC1A, exhibits more than 250× shorter transition latency than the existing deep package C-state PC6.

While we demonstrate APC potential for Intel servers, which account for more than 80% of the entire server processor market [18], our proposed techniques are general, hence applicable to other server processor architectures.

In summary, this work makes the following contributions:

- APC is the first practical package C-state design targeting the killer microseconds problem in datacenter servers running latency-critical applications.
- APC introduces the PC1A low-power package C-state that a system can enter once all cores enter a shallow C-state (i.e., CC1).
- APC improves existing deep package C-states by drastically reducing their transition latency (>250×) while retaining a significant fraction of their power savings.
- Our evaluation shows that APC reduces the energy consumption of Memcached by up to 41% with less than 0.1% performance degradation. APC achieves similar gains for other representative workloads.

2. MOTIVATION

Modern servers running latency-critical applications are stuck in PC0 (i.e., active package C-state) and never enter PC6, because CC6 is disabled in these systems [53, 54, 57]. A major consequence of this is that the server experiences high power consumption from the uncore components in the processor SoC (e.g., last-level-cache, IO interfaces) and DRAM, which are always active [26]. Our measurements (see Sec. 6) of an idle system (all cores in CC1) show that uncore & DRAM power consumption accounts for more than 65% of the SoC & DRAM power consumption.

Adding a deep agile package C-state PC1A that 1) has a sub-microsecond transition time and 2) only requires cores to enter CC1 would significantly improve energy proportionality for servers by drastically reducing uncore and DRAM power consumption when all cores are idle. Eq. 1 estimates the power savings that PC1A C-state could bring.

\[
P_{\text{baseline}} = R_{PC0} \times P_{PC0} + R_{PC0\text{idle}} \times P_{PC0\text{idle}}
\]

\[
\%P_{\text{savings}} = \frac{R_{PC1A} \times (P_{PC0\text{idle}} - P_{PC1A})}{P_{\text{baseline}}} \tag{1}
\]

\[P_{\text{baseline}}\] is the overall, SoC & DRAM, power of a current server measured as the sum of the power while the system has at least one core in CC0 and when all cores are idle in CC1 (i.e., \(P_{PC0}\) and \(P_{PC0\text{idle}}\)) weighted by their respective state residencies \(R_{PC0}\) and \(R_{PC0\text{idle}}\). We can obtain the savings of PC1A from Eq. 1 by using the power of the new proposed state \(P_{PC1A}\) (shown in Table 1 and derived in Sec. 5) and assuming that the fraction of time a server will spend in PC1A is the same as the time the baseline spends in PC0idle (i.e., \(R_{PC1A} = R_{PC0\text{idle}}\)).

For example, we consider running a key-value store workload (e.g., Memcached [49]) on a processor with 10 cores. Our experimental analysis (see Sec. 6) reveals that all cores
3. BACKGROUND

Fig. 1(a) shows the floorplan for an Intel Skylake Xeon server processor (SKX), consisting of three major building blocks: the mesh tiles, north-cap, and DDR IOs (PHYS). SKX uses a mesh network-on-chip to connect cores, memory controllers (MC), and IO controllers (North Cap) [82, 83, 85].

Core tiles. The largest area contributor to the entire SoC area are the core tiles (Fig. 1(b)). Each of which contains 1) all core domain (CPU core, AVX extension, and private caches) and 2) a portion of the uncore domain (caching-and-home-agent (CHA), last-level-cache (LLC), and a snoop filter (SF)) [82].

North-Cap. The top portion of the SoC die is called the north-cap [82, 83]. It consists of the high-speed IO (PCIe, UPI, and DMI) controllers and PHYs, serial ports, fuse unit, clock reference generator unit, and the firmware-based global power management unit (GPMU).

Power Delivery Network (PDN). The PDN is the SoC subsystem responsible for providing stable voltage to all the processor domains [12, 30, 37, 67, 82]. Fig. 1(c) shows the organization of the SoC into voltage domains. SKX implements [82] nine primary voltage domains generated using a FIHV (fully integrated voltage regulator [12, 37, 67, 82]) or MBVR (motherboard voltage regulator [25, 29, 75]). For example, each core has a dedicated FIVR (Vcc core), and the CLM (CHA, LLC, mesh interconnect) has two FIVRs (Vccclm0 and Vccclm1); IO controllers and PHYs use MBVR (Vccsa and Vecio, respectively) [82].

Clock Distribution Network (CDN). A CDN distributes the signals from a common point (e.g., clock generator) to all the elements in the system that need it. Modern processors use an all-digital phase-locked loop (ADPLL) to generate the CPU core clock [82]. An ADPLL maintains high performance with significantly less power as compared to conventional PLLs [25]. SKX system uses multiple PLLs: a PLL per core [82], a PLL per each high-speed IO (i.e., PCIe, DMI, and UPI controller) [39], one PLL for the CLM domain [82], and one PLL for the global power management unit [83].

3.1 Power Management States

Power management states reduce power consumption while the system or part of it is idle. Modern processors support multiple power states such as Core C-states, IO link-state (L-state), DRAM power mode, and Package C-state.

Core C-states (CCx). Power saving states enable cores to reduce their power consumption during idle periods. We refer to core C-states as CCx; CC0 is the active state, and higher values of x correspond to deeper C-states, lower power, and higher transition latency. For example, the Intel Skylake architecture offers four core C-states: CC0, CC1, CC1E, and CC6 [26, 28, 78]. While C-states reduce power, a core cannot be utilized to execute instructions during the entry/exit to/from a C-state. For example, it is estimated that CC6 requires 133µs transition time [45, 46]. As a result, entry/exit latencies can degrade the performance of services that have microseconds processing latency, such as in user-facing applications [49].

IO L-states (Lx). High-speed IOs (Links) support power states that provide similar performance/power trade-offs to core C-states [26]. While specific power states differ based on the type of link, the high-level concepts we describe here are similar. L0 is the active state, providing maximum bandwidth and minimum latency. L0s is a standby state, during which a subset of the IO lanes are asleep and not actively transmitting data. The reference clock and internal PLLs are kept active to allow fast wakeup (typically <64µs [26, 38, 41]) while providing significant (up to ~50% of L0) power savings. L0p is similar to L0s state, but a subset of the data lanes remain awake (typically half). Bandwidth is reduced, and latency for transmitting data increases. L0p provides up to ~25% lower power than L0 with faster exit latency than L0s (typically ~10ns). The IO link-layer autonomously handles the entry to L0s/L0p states (no OS/driver interactions) once the IO link is idle [26]. L1 is a power-off state, meaning that the link must be retrained, and PLLs must be switched on to resume link communication. L1 provides higher power saving than L0s and L0p but requires a longer transition latency (several microseconds).

DRAM Power Saving Techniques. Modern systems im-
implement two main DRAM power-saving techniques: CKE modes and self-refresh [6, 19, 26, 64].

CKE modes: CKE (clock enable) is a clock signal the memory-controller (MC) sends to the DRAM device. When the MC turns-off the CKE signal, the DRAM can enter low power modes. There are two main types of CKE power-modes in DDR4: 1) Active Power Down (APD), which keeps memory pages open and the row buffer powered on, and 2) Pre-charged Power Down (PPD), which closes memory pages and powers down the row buffer. The granularity of CKE modes is per rank and it is considered a relatively quick technique (independent of the power mode used), with nanosecond-scale transition latency (10ns – 30ns) and significant power savings (≥50% lower power than active state) [6, 19, 64].

Self-refresh: In system active state, the MC is responsible to issue the refresh commands to DRAM. To reduce power consumption in MC and DRAM device, DRAM support a self-refresh mode, in which the DRAM is responsible for the refresh process. Once the MC places the DRAM in Self-refresh mode, the power management unit can turn off the majority of the interface between the SoC and the DRAM [31]. Due to this deep power-down, the exit latency of self-refresh is several microseconds. To minimize the performance impact of self-refresh exit latency, the power management unit of modern processors allow transitions to the self-refresh state only while in a deep idle power state (e.g., package C-states) [6, 26, 28].

| PCx | Cores in CCx | L3 Cache | PLLs | PCIe/DMI | UPI | DRAM |
|-----|--------------|----------|------|----------|-----|------|
| PC0 | >1 in CC0    | Accessible | On   | L0       | L0  | Available |
| PC6 | All in CC6   | Retention | Off  | L1       | L1  | Self Refresh |
| PC1A| All in CC1   | Retention | On   | L0s      | L0p | CKE off |

Table 2: SKX package C-state characteristics and our new PC1A (details in Sec. 4).

Package C-states (PCx). Package C-states are responsible for reducing power consumption of the uncore and other system components (e.g., DRAM) once all cores are idle. Skylake servers support three main package C-states: PC0, PC2, PC6 [26, 78]. PC0 is the active state, enabled when at least one core is active (i.e., in CC0). PC2 is a non-architectural (intermediate) state, which is typically used as a transient state between PC0 and deeper package C-states. PC6 is a deep package C-state that saves significant uncore power by placing the IOs in low power modes and reducing the CLM voltage to retention, as shown in Table 2. However, PC6 has high (≥50us, see Table 1) transition latency. Due to its high transition latency, server vendors typically recommended disabling PC6 to prevent response time degradation [53, 54, 57].

![Figure 3: Main APC architecture components (in color).](image)

4. AgilePkgC (APC) ARCHITECTURE

The main APC components introduced to implement the new PC1A package C-state are shown in Fig. 3. This architecture is based on three main components: 1) the Agile Power Management Unit (APMU), 2) the IO Standby Mode (IOSM), and 3) the CHA, LLC, and Mesh Retention (CLMR), discussed in Sec. 4.1, Sec. 4.2, and Sec. 4.3, respectively. APMU triggers PC1A system-level power management flow once all cores enter the CC1 shallow C-state (see Table 2) and requires additional signals, red in Fig. 3, to interface with the existing, firmware-based global PMU (GPMU). IOSM enables power saving in the IO domain (i.e., PCIe, DMI, UPI, DRAM) by exploiting IO shallow low-power modes and requires adding specific signals depicted in blue, orange, and purple in Fig. 3. CLMR enables power savings in the CLM domain and requires adding two signals to CLM’s FIVRs and one to CLM’s clock tree, shown in green and brown in Fig. 3.

We first describe the APMU and the PC1A transition flows that it implements, then we describe in detail the IOSM (Sec. 4.2) and CLMR (Sec. 4.3) components PC1A uses.

4.1 Agile Power Management Unit (APMU)

APC introduces APMU to enable system-level power savings by entering PC1A with nanosecond-scale transition latency. This innovation involves agile coordination of multiple SoC domains (e.g., CPU cores, high-speed IOs, CLM, DRAM). Whereas, rather than trying to enter domain’s deep power states (e.g., core CC6, PCIe L1, DRAM self-refresh), PC1A leverages shallower power states (e.g., core CC1, PCIe L0s, DRAM CKE-off) and enables significant power savings.
with a nanosecond-scale exit latency. Particularly, APMU orchestrates the \textit{PC1A} flow by interfacing with five key SoC components (as shown in Fig. 3): 1) CPU cores, 2) high-speed IOs (PCIe, DMI, and UPI), 3) memory controller, 4) CLM FIVR and clock tree, and 5) global PMU (GPMU).

We place the APMU in north-cap, close to the firmware-based GPMU and IO domain [82, 83]. APMU implements three key power management infrastructure components. First, a hardware fast (nanosecond granularity) \textit{finite-state-machine} ( FSM) that orchestrates \textit{PC1A} entry and exit flows. The APMU FSM uses the same clock as the GPMU.

Second, \textit{status and event signals} that feed into the APMU FSM. The InCC1 status signal combines (through AND gates) the status of all cores to notify the APMU that all cores are in the \textit{CC1} power state. Similarly, the InL0s status signal notifies the APMU that all IOs are in \textit{L0s} power state (see Sec. 4.2). The GPMU \textit{WakeUp} signal sends a wakeup event to the APMU when an interrupt (e.g., timer expiration) occurs. The \textit{PwrOk} signal notifies the APMU when the CLM FIVR reaches its target operational voltage level after exiting retention mode (see Sec. 4.3).

Third, APC implements \textit{control signals} that the APMU uses to control APC components. The \textit{Allow_CKE_OFF} control signal, when set, enables the MC to enter \textit{CKE off} low power state and to return to active state when unset. Similarly, the \textit{AllowL0s} signal, when set, enables the IO interfaces to enter \textit{L0s} power state and to return to active state when unset (see Sec. 4.2). When Ret signal is set, the CLM FIVRs reduce their voltage to pre-programmed retention level and they restore the previous voltage level when Ret is unset (see Sec. 4.3). The APMU notifies the GPMU that the system in \textit{PC1A} by setting the InPC1A signal.

\textbf{PC1A Entry and Exit Flows.} APC power management flow, implemented by the APMU, is responsible for orchestrating the transitioning between \textit{PC0} and \textit{PC1A}, as depicted in Fig. 4.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{fig4.png}
\caption{Power management flow for the \textit{PC1A} C-state.}
\end{figure}

The \textit{PC1A} flow first transitions from \textit{PC0} to an intermediate state, \textit{ACC1}, as soon as all cores enter \textit{CC1}. \textit{ACC1} serves the purpose of setting the \textit{AllowL0s} signal, thus allowing IOs to enter the \textit{L0s} state. Next, once all IOs in \textit{L0s} state (\&InL0s) the flow performs two branches, (i) and (ii), concurrently: (i) it clock-gates the CLM and (ii) initiates a non-blocking CLM voltage transition (by setting the Ret signal) to reduce the voltage to retention level. (ii) it sets \textit{Allow_CKE_OFF} to allow the MCs to enter \textit{CKE off}. In contrast to existing package C-states (e.g., \textit{PC6}, shown in Fig. 2), the flow keeps all system phase-locked loops (PLLs) powered-on. After these two steps (2) is non-blocking) the system is in the \textit{PC1A} C-state.

Exiting \textit{PC1A} can happen because of two main causes: First, an IO link generates a \textit{wake up event} when exiting \textit{L0s} due to traffic arrival; as soon as the link starts the transition from \textit{L0s} to \textit{L0}, the IO link unsets \textit{InL0s}, generating a \textit{wake up event} in the APMU. Second, the GPMU generates an explicit \textit{wake up event} by setting the \textit{WakeUp} signal. The GPMU generates a \textit{wake up event} for multiple reasons, such as an interrupt, timer expiration, or thermal event.

When a \textit{wake up event} occurs, the system \textit{exits} the \textit{PC1A} by reversing the entry flow in two branches, (i) and (ii), concurrently: (i) it unsets the Ret signal to ramp up the CLM voltage to its original level; when the FIVRs set PwrOk, (ii) the flow clock-uncugates the CLM. (ii) it unsets \textit{Allow_CKE_OFF} to reactivate the MCs. Once both branches are completed, the flow reaches the \textit{ACC1} state. Finally, in case the \textit{wake up event} is a \textit{core interrupt}, the interrupted core transitions from \textit{CC1} to \textit{CC0}, correspondingly transitioning the system from \textit{ACC1} state to \textit{PC0} active state. At this step, the flow unsets \textit{AllowL0s} to bring the IO links back to the active \textit{L0} state.

\section{IO Standby Mode (IOSM)}

IOSM leverages IO shallow power states (e.g., \textit{L0s}, \textit{CKE off}) to enable significant power savings in \textit{PC1A} with a nanosecond-scale exit latency. We discuss PCIe, DMI, and UPI shallow power states in Sec. 4.2.1 and DRAM shallow power mode in Sec. 4.2.2.

\subsection{PCIe, DMI, and UPI in Shallow Power States}

Once an IO interface is idle (i.e., not sending or receiving any transaction), the IO link and controller can enter to idle power state, called \textit{L-state}, as explained in Sec. 3.1. Deep \textit{L-states (L1)} have an exit latency of several \textmu s, making them unsuitable for APC. Instead, we allow links to enter the \textit{L0s} state, which has exit latency in the order of tens of nanoseconds (e.g., 64\textmu s). While \textit{L0s} could be entered while other agents are active, datacenter servers normally completely disable it to avoid performance degradation [53, 54, 57]. For the same reason, APC keeps \textit{L0s} disabled when cores are active and allows high-speed IOs (e.g., PCIe, DMI, and UPI) to enter \textit{L0s} only when all the cores are idle (i.e., all cores in \textit{CC1}).

\textbf{AllowL0s Signal.} To only allow entering \textit{L0s} when all cores are idle, APC requires a new signal, \textit{AllowL0s} (light blue in Fig. 3), to each IO controller. The power management \textit{sets} the signal once all cores are in \textit{CC1} and each IO controller autonomously initiates the entry to \textit{L0s} state once the IO link is idle (i.e., no outstanding transaction) [26]. To allow the IO controller to enter quickly to \textit{L0s} once the IO link is idle, the \textit{AllowL0s} signal also sets the \textit{L0s entry latency} (L0s\_ENTRY\_LAT [41]) configuration register. Sets\textsuperscript{3}\textsuperscript{4} PCIe, DMI, and UPI (UPI’s previous generation) support \textit{L0s}. UPI support \textit{L0p} state [26, 57, 69]. We allow the IO to enter to \textit{L0p} in case it does not support \textit{L0s}. \textit{L0p} exit latency is \textasciitilde{10}\textmu s [26].

\textsuperscript{3}When \textit{L0s} is enabled and the link is in the \textit{L0} state, the IO controller transitions the link to the \textit{L0s} state if idle conditions are
ting L0s_ENTRY_LAT to “1” sets the entry latency to 1/4 of the L0s exit latency, which is typically <64ns [38, 41]).

InL0s Indication. In the baseline system, the IO link power status (i.e., L0, L0s, and L1) is stored in a register inside the IO controller [43]. Therefore, when entering a package C-state, the power management firmware needs to read this register. To make the new PC1A agile, we add an output signal, InL0s (orange in Fig. 3), to each one of the high-speed IO controllers. The IO link layer sets the signal if the IO is at L0s or deeper and unsets it if the link is in active state (i.e., L0) or is exiting the idle state. The IO controller should unset the signal once a wakeup event is detected to allow the other system components to exit their idle state during PC1A exit flow concurrently; this transition only requires tens of nanoseconds.

4.2.2 DRAM in a Shallow Power State

When entering existing deep package C-states (e.g., PC6), the flow allows the memory controller to put DRAM into self-refresh mode (as shown in Fig. 2). The exit latency from self-refresh mode is several microseconds (see Sec. 3.1) and unsuitable for PC1A.

Allow_CKE_OFF Signal. Instead of using the long latency self-refresh mode, APC instructs the memory controller (MC) to put DRAM into CKE off mode, which has lower power savings compared to self-refresh mode but massively lower exit latency (<30ns). To enable this transition, APC adds a new input signal, Allow_CKE_OFF to each memory controller (purple in Fig. 3). When this signal is set, the memory controller enters CKE off mode as soon as it completes all outstanding memory transactions and returns to the active state when unset.

4.3 CHA, LLC, and Mesh Retention (CLMR)

In our reference, skylake-based multicore design, the last-level cache (LLC) is divided into multiple tiles, one per core, as Fig. 1(b) and Fig. 3 illustrate. Each tile includes a portion of the LLC memory, a caching and home agent (CHA) and a snooping filter (SF); a mesh network-on-chip (NoC) connects the tiles with the IOs and memory controllers (MCs) [82]. Two FIVR voltage domains (VccCM0 and VccCM1) power the CHA, LLC, and the (horizontal) mesh interconnect (known as CLM), as illustrated in Fig. 1(c). When entering existing deep package C-states (i.e., PC6), the GPMU firmware turns off the phase-locked loop (PLL) for the CLM and reduces the VccCM voltage to retention level to reduce leakage power. During PC6 exit, the firmware 1) sends messages to the FIVRs to ramp up the VccCM voltage and 2) re-locks the PLL (few microseconds).

To cut the time of re-locking the CLM PLL, APC keeps the PLL locked and uses a new ClnkGate signal (brown in Fig. 3) to allow quickly clock gating CLM’s clock distribution network (e.g., clock tree). To allow agile power management response, APC adds a new signal, Ret to each CLM FIVRs (green in Fig. 3). When Ret is set, the two CLM FIVRs reduce the voltage to pre-programmed retention voltage; when Ret is unset, the FIVRs ramp their voltage back to the previous operational voltage level. Once the FIVR voltage level reaches the target, the FIVR sets the Power signal.

5. IMPLEMENTATION AND HW COST

APC requires the implementation of three main components: the IOSM subsystem, the CLMR subsystem, and the agile power management unit (APMU). We discuss implementation details for each component, including area and power cost, and the transition latency for the new PC1A state.

5.1 IO Standby Mode (IOSM)

IOSM requires the implementation of three signals depicted in Fig. 3: 1) AllowL0s (light blue), 2) InL0s (orange), and 3) Allow_CKE_OFF (purple).

Implementing AllowL0s requires routing control signals from the APMU to each one of the high-speed IO controllers (i.e., PCIe, DMI, and UPI). In each IO controller, the AllowL0s signal control overrides the control register (e.g., LNKCON.active_state_link_pm_control [42]) that prevents the Link Training and Status State Machine (LTSSM) from entering L0s when the IO link is idle [11, 13, 66]. We implement InL0s using the LTSSM status: the IO controller sets InL0s once the LTSSM reaches the L0s state and unset it once the LTSSM exits the L0s (i.e., a wakeup event is detected). The InL0s output of each IO controller is routed to the APMU. To reduce routing overhead, the InL0s of neighbouring IO controllers are aggregated using AND gates and routed to the APMU, as shown in Fig. 3.

Similarly, implementing Allow_CKE_OFF requires routing a control signal from the APMU to each of the two memory controllers, as shown in Fig. 3. The Allow_CKE_OFF control signal overrides the control register in the memory controller (e.g., MC_INIT_STAT_C.cke_on [42]) that prevents an idle memory controller entering CKE off mode.

Overall, IOSM adds few long distance signals. In comparison to the number of data signals in an IO interconnect (mesh or ring), which typically has 128-bit – 512-bit data width [5, 24], the additional five signals represent 1 – 4% extra IO interconnect area. We extrapolate the IO interconnect area from a SKX die. The IO interconnect in north-cap [82] is less than 6% of SKX die area. Thus, the area overhead of the five new signals is <0.24%/<0.06% of SKX die area (assuming 128-bits/512-bits IO interconnect width). This is a pessimistic estimate, since the IO interconnect includes control signals in addition to data.

Implementing the additional signals in the high-speed IOs (i.e., AllowL0s and InL0s), and the memory (i.e., Allow_CKE_OFF) controllers only requires small modifications, since the required control/status knobs/signals are already present in the controllers. Based on a comparable power-management flow implemented in [31], we estimate the area required to implement the signals to be less than 0.5% of each IO controller area. Given that the IO controllers take

---

3In case that no device is present, the link power state is known as: no device attached (NDA), which is deeper than L1 state [35].

4The Vecio (Fig. 1[c]) fixed voltage domain, delivered from a motherboard voltage regulator, powers the vertical mesh [82].

5To maximize performance, vendors recommend to disable IO power management and link state, including L1 and L0s [53, 54, 57].

6The LTSSM FSM manages the link operation of each high-speed IO [11, 13, 66].
InCC, we estimate the area required for the PC1A controller to be smaller than the existing CLM clock-tree control logic. To control the CLM FIVRs voltage, we route an additional control signal, ClkGate, from the APMU to the two FIVRs that power the CLM [82]. To enable a FIVR to directly transition to a pre-programmed retention voltage, we add to each FIVR control module (FCM [12, 67]) an 8-bit register that holds the retention voltage identification (RVID) value [63, 74]. Finally, we add a PowErr status signal that the FIVR uses to notify the APMU that the voltage is stable. Overall, CLMR adds three long distance signals. Using analogous analysis as in Sec. 5.1, we estimate the area overhead for the three new signals is <0.14% of SKX die area. To implement the new RVID 8-bit register in each FIVR’s FCM and add a new logic to select between the RVID and the original VID, needs less than 0.5% of the FCMs’ area. The FIVR area is less than 10% of the SKX core die area and a core area in a die with 10 cores will be less than 10% of the original VID, needs less than 0.005%.

5.2 CHA, LLC, and Mesh Retention (CLMR)

Implementing CLMR requires two main components 1) CLM clock-tree gating and 2) CLM voltage control. To allow clock gating/ungating of the CLM clock-tree, we route a control signal ClkGate from the APMU to the existing CLM clock-tree control logic. To control the CLM FIVRs voltage, we route an additional control signal, ClkGate, from the APMU to the two FIVRs that power the CLM [82]. To enable a FIVR to directly transition to a pre-programmed retention voltage, we add to each FIVR control module (FCM [12, 67]) an 8-bit register that holds the retention voltage identification (RVID) value [63, 74]. Finally, we add a PowErr status signal that the FIVR uses to notify the APMU that the voltage is stable. Overall, CLMR adds three long distance signals. Using analogous analysis as in Sec. 5.1, we estimate the area overhead for the three new signals is <0.14% of SKX die area. To implement the new RVID 8-bit register in each FIVR’s FCM and add a new logic to select between the RVID and the original VID, needs less than 0.5% of the FCMs’ area. The FIVR area is less than 10% of the SKX core die area and a core area in a die with 10 cores will be less than 10% of the SoC area, so the overall area overhead of two FCMs is negligible (less than 0.005%).

5.3 Agile Power Management Unit (APMU)

The APMU, is implemented using a simple finite-state-machine (FSM) connected to the global PMU (GPMU), as depicted in Fig. 3. APMU monitors its input status signals and drives its control signals as shown in Fig. 4. Based on a comparable power-management flow implemented in [31], we estimate the area required for the PC1A controller to be up to 5% of the GPMU area. As shown in Fig. 1 (dark blue), the GPMU area is less than 2% of the SKX die area. Therefore, APMU area is less than 0.1% of the SKX die area.

We also need to implement a global status signal, InCC1, that determines when all the CPU cores are at CC1 power state. The power state of each core is known to each core’s power management agent (PMA [76]), therefore, we simply expose this status as an output signal from each CPU core. The InCC1 output of each CPU core is routed to the APMU. To save routing resources, the InCC1 of neighbouring cores are combined with AND gates and routed to the APMU, as shown in blue in Fig. 3. In total we have three long distance signals; according to our analysis in Sec. 5.1, their area overhead is <0.14% of the SKX die area.

In summary, the three APC components discussed in Sections 5.1, 5.2, and 5.3 incur <0.75% overhead relative to a SKX die area.

5.4 PC1A Power Consumption Analysis

To estimate the PC1A power, we carry out multiple measurements of our reference system (configuration in Sec. 6) to isolate the individual components contributing to the PC1A power consumption. As shown in Table 2, the power consumption difference between PC1A and PC6 is due to the: 1) CPU cores (\(P_{\text{cores}, \text{diff}}\)), 2) IOs power (\(P_{\text{IOs}, \text{diff}}\)), 3) PLLs (\(P_{\text{PLLs}, \text{diff}}\)), and 4) DRAM (\(P_{\text{DRAM}, \text{diff}}\)). Therefore, the PC1A SoC power, \(P_{\text{soc}, \text{PC1A}}\), can be estimated as in Eq. 2.

\[
P_{\text{soc}, \text{PC1A}} = P_{\text{soc, PC6}} + P_{\text{cores, diff}} + P_{\text{IOs, diff}} + P_{\text{PLLs, diff}}
\]

Similarly, the PC1A DRAM power consumption, \(P_{\text{DRAM, PC1A}}\), can be estimated as in Eq. 3:

\[
P_{\text{DRAM, PC1A}} = P_{\text{DRAM, PC6}} + P_{\text{DRAM, diff}}
\]

We use Intel’s RAPL monitoring interface [23, 27, 55] to measure the SoC (package) and DRAM power consumption. Next, we discuss the two configurations we use to determine each one of the four power deltas between PC1A and PC6.

\(P_{\text{cores, diff}}\): To measure the cores power difference between our new PC1A and PC6, denoted by \(P_{\text{cores, diff}}\). We use two system configurations: 1) all cores are placed in CC1 and 2) all cores are placed in CC6. To keep uncore power consumption similar in the two configurations, we disable uncore power savings techniques such as package C6, DRAM opportunistic self-refresh (OSR), memory-power-down (CKE off), uncore frequency scaling [20, 26, 57]. We measure the power of the two configurations using RAPL.Package [23, 27, 55] and calculate the difference. Our measurements shows that \(P_{\text{cores, diff}} \approx 12.1\) W.

\(P_{\text{IOs, diff}}\) and \(P_{\text{DRAM, diff}}\): The IOs power includes PCIe, DMI, UPI, and memory controllers and their corresponding physical layers (PHYs) but it does not include the devices’ (e.g., DRAM) power. To measure the IOs power consumption difference between PC1A and PC6, denoted by \(P_{\text{IOs, diff}}\), we use two configurations: 1) place the PCIe and DMI in L0s power state, UPI in L0p power mode, and memory-controller (MC) in CKEoff power mode and 2) place the PCIe, DMI, and UPI in L1 power state, and memory-controller (MC) in self-refresh power mode. To place the system in these power modes, we use BIOS configurations to i) place the cores in core CC6 and set the package C-state limit to PC2 to allow the IOs to enter to local power mode but prevent the system from entering PC6 [34], ii) set the PCIe/DMI/UIPI active state power management to L0s/L0p/L0d for the first configuration and to L1/L1/L1 for the second configuration [57], and iii) configure the memory to enter power-down (CKE off) and opportunistic self refresh (OSR) [20, 26, 57] for the first and second configuration, respectively. To obtain \(P_{\text{IOs, diff}}\) (\(P_{\text{DRAM, diff}}\)) we measure the power of the two configurations using RAPL.Package (RAPL.DRAM) [23, 27, 55] and calculate the difference. Our measurements shows that \(P_{\text{IOs, diff}} \approx 3.5\) W and \(P_{\text{DRAM, diff}} \approx 1.1\) W.

\(P_{\text{PLLs, diff}}\): All PLLs are on in PC1A, but off in PC6. We estimate the PLLs power consumption difference between our new PC1A and PC6, denoted by \(P_{\text{PLLs, diff}}\), by: number of system PLLs times a PLL power. In our SKX system [36] there are approximately 18 PLLs: one PLL for each PCIe, DMI, and UPI controller [39] (our system [36] has 3 PCIe, 1 DMI, and 2 UPI), one PLL for the CLM and memory controllers [82], one PLL for the global power management unit [83], and one PLL per core (10 cores in our system [36]). The per core PLL power is accounted for in \(P_{\text{cores, diff}}\), since we measure RAPL Package. Therefore, there are 8 remaining PLLs. The Skylake system uses all-digital phase-locked
When an interrupt occurs, the system moves back to PC. The wake-up event is signaled to the APMU; when all the cores enter to CC (i.e., $L_1$) (0.51W), respectively. In summary, $P_{soc, PCA} \approx 11.9W + 12.1W + 3.5W + 0.057W \approx 27.5W$ and $P_{dram, PCA} \approx 0.51W + 1.1W \approx 1.6W$, as we summarize in Table 1.

### 5.5 PC1A Latency

We estimate that the overall transition time (i.e., entry followed by direct exit) for the APC’s PC1A state to be <200ns: >250× faster than the >50μs that PC6 requires. Next, we discuss in detail the entry and exit for PC1A; we refer to the power management flow shown in Fig. 4.

#### 5.5.1 PC1A Entry Latency

The package C-state flow starts once all cores are idle; when all the cores enter to CC1, the system transitions to ACC1 package state. Similar to the traditional PC2 package C-state (shown in Fig. 2, ACC1 is a temporary state at which uncore resources (LLC, DRAM, IOs) are still available. Therefore, we measure PC1A latency starting from the ACC1.

In ACC1, we enable the IOs to enter a shallow power state (i.e., $L_0$s). As discussed in Sec. 4.2.1, the entry latency of the IO (PCIe, DMI, and UPI) controllers is ≈ 25% of the exit latency (typically <64ns). Therefore, once the IOs are idle for 16ns the IO enters $L_0$s state and sets the $T_{L_0s}$ signal. In case some IOs are not idle, the system remains in ACC1. When an interrupt occurs, the system moves back to PC0.

Clock-gating the CLM domain and keeping the PLL ON typically takes 1 – 2 cycles in an optimized clock distribution system [22, 79]. Reducing CLM’s voltage from nominal voltage (~0.8V) to retention voltage (~0.5V) [1, 14], is a non-blocking process. FIVR’s voltage slew rate is typically $\geq 2mV/ns$ [12, 51]. Thus, the time it takes for the FIVR to reduce the voltage by 300mV (from ~0.8V to ~0.5V) is ≤150ns. Asserting MCs’ Allow_CKE_OFF control signal takes 1 – 2 cycles. Since the system is idle, once the MCs receive the $Allow\_CKE\_OFF$ signal they enter CKE off mode within 10ns [19, 64].

In summary, since voltage transition to retention and entry to CKE off mode are non-blocking, PC1A entry latency is ~18ns using a power management controller with 500MHz clock frequency.\(^\text{10}\)

#### 5.5.2 PC1A Exit Latency

PC1A exit is caused by wakeup events (e.g., IO activity, GPMU timer). In case of IO events, the IO links concurrently start exiting $L_0$s to $L_0p$ (a process that requires <64ns) and a wake-up event is signaled to the APMU.

Increasing the CLM’s voltage from retention (~0.5V) to nominal voltage (~0.8V) [1, 14], takes 150ns since FIVR’s voltage slew rate is typically $\geq 2mV/ns$ [12, 51].\(^\text{11}\)Clock-gating the CLM domain and keeping the PLL ON typically takes 1 – 2 cycles in an optimized clock distribution system [22, 79]. Unsetting MCs’ $A1\_low\_CE\_OFF$ control signal takes 1 – 2 cycles. Once the MCs receive the $A1\_low\_CE\_OFF$ signal, they exit MCs CKE off mode within 24ns [6, 19, 64].

In summary, PC1A exit latency is ≤150ns using a power management controller with 500MHz clock frequency. The worst case entry plus exit latency is ≤168ns. We conservatively assume ≤200ns.

### 5.6 Design Effort and Complexity

APC proposed techniques involve non-negligible front-end and back-end design complexity and effort. The APMU, PC1A control flows, IOSM, and CLMR, require careful pre-silicon verification to ensure that all the hardware flows (described in Fig. 4). IO controllers (PCIe, DMI, UPI, MC), and CPU core changes operating as expected by the architecture specification. The effort and complexity can be significant due to two main reasons. 1) APC involves system-on-chip global changes, requiring careful coordination between multiple design teams. 2) the power management flows are hardware-based, which, compared to firmware-based flow, reduces the opportunity to patch the flows if a hardware bug is found post-silicon production.

However, APC effort and complexity are comparable to recent techniques implemented in modern processors to increase their energy efficiency (e.g., hybrid cores [77, 84]). Therefore, we believe that once there is a strong request from customers and/or pressure from competitors, several processor vendors will eventually implement a similar architecture to APC to increase server energy efficiency significantly.

### 6. EXPERIMENTAL METHODOLOGY

We evaluate APC using three latency-critical services: Memcached, Apache Kafka, and MySQL. Memcached [2] is a popular key-value store commonly deployed as a distributed caching layer to accelerate user-facing applications [68, 72, 87]. Memcached has been widely studied [58, 60, 73, 86], particularly for tail latency performance optimization [7, 59, 68]. Kafka [56] is a real-time event streaming platform used to power event-driven microservices and stream processing applications. MySQL [70] is a widely used relational database management system.

We use a small cluster of servers to run our three services and the corresponding clients. Each server has an Intel Xeon Silver 4114 [36] processor running at 2.2 GHz nominal frequency (minimum 0.8 GHz, maximum Turbo Boost frequency 3 GHz) with 10 physical cores (total of 20 hyper-threads) and 192 GB of ECC DDR4 2666MHz DRAM.

**Workload setup.** For each of our three services (Memcached, Kafka, MySQL), we run a single server process on a dedicated machine and corresponding clients on separate machines. We pin server processes to specific cores to minimize the impact of the OS scheduler. The Memcached client is a modified version of the Mutilate load generator [58] set to reproduce the ETC Facebook workload [8] using one master and four workload-generator clients, each running on a separate machine. The Kafka client consists of the Consumer-need to exit in the middle of a voltage transition to retention) [69].
Performance and ProducerPerformance Kafka. The MySQL client consists of the sysbench benchmarking tool using the OLTP test profile [4].

**Baseline configurations.** We consider two baseline configurations: $C_{\text{shallow}}$ and $C_{\text{deep}}$. The $C_{\text{shallow}}$ configuration is representative of real modern datacenters that, as discussed in Sec. 1, are normally configured for maximum performance [53, 54, 57]. Therefore, in the $C_{\text{shallow}}$ configuration, we disable the CC6 and CC1E core C-states and all package C-states. Additionally, we disable P-states (i.e., DVFS) by setting the frequency scaling governor to *performance* mode (i.e., nominal frequency), to avoid frequency fluctuations. The $C_{\text{deep}}$ configuration has all core and package C-states enabled. P-states are still disabled, but the frequency scaling governor is set to *powersave* mode. In order to allow the system to enter PC6, we tune it using the auto-tune option from *powertop* [3]. We obtain C-state residency and number of transitions using residency reporting counters [40], and we use the RAPL interface [33] to measure power consumption.

**Power and performance models.** We estimate the impact of the APC on power and performance with a combination of simple models and real measurements. We base power estimations on the same model as in Eq. 1 (Sec. 2). For the performance model, we calculate the impact on average latency by combining the number of PC1A transitions, measured on our baseline system, with the additional transition latency required for PC1A (see Sec. 5.5).

**Power event tracing.** We estimate the opportunity for PC1A residency using Intel’s SoCWatch [44] energy analysis collection tool. We use SoCWatch to generate a trace that records C-state transition events, and we process this timeline to identify opportunities to enter PC1A. Due to sampling constraints, SoCwatch does not record idle periods shorter than 10 us; therefore, the PC1A opportunity we present in Sec. 7 underestimates the real opportunity. We additionally use SoCWatch to measure the distribution of the number of active cores after full idle periods (i.e., periods during which all cores are in CC1 or lower C-state). We use this metric and the PC1A transitions to estimate the performance impact presented in Sec. 7.

7. **EVALUATION**

Our evaluation of APC addresses the following questions:

1. What is the opportunity to enter APC’s new agile deep package C-state (PC1A)?
2. What are the power savings PC1A can enable?
3. How does PC1A impact performance?

We first focus on the Memcached [2] service and later discuss results on our two other workloads in Sec. 7.4. We tune the client to generate a wide range of request intensity, but focus on the lower end (approximately $5 - 20\%$ processor utilization), which represents the typical operating range of servers running latency-critical applications [47, 62, 91–94]. For Memcached, this load range corresponds to a range of $4K - 100K$ QPS (queries per second). In our plots, we highlight the low-load region with a shaded area.

First, we analyze the performance impact of enabling deep core C-states and package C-states on Memcached. Fig. 5 compares the average and tail latency of Memcached running on the $C_{\text{shallow}}$ configuration against the $C_{\text{deep}}$ configuration (which enables deep C-states, see Sec. 6).

The $C_{\text{shallow}}$ configuration has significantly better average and tail latency compared to the $C_{\text{deep}}$ configuration, as it avoids deep core C-state transition overhead, thus corroborating the advise of server manufacturers. However, the $C_{\text{shallow}}$ configuration also prevents entering any power-saving package C-state, thus missing the opportunity to save package power during periods of full system idleness. At high load ($\geq 300K$ QPS) of the $C_{\text{deep}}$ configuration, we observe a latency spike caused by CC6/PC6 transitions delaying the processing of the initial incoming requests, which further delays and queues following requests.

7.1 **PC1A Opportunity**

Fig. 6 quantifies the opportunity for the system to enter APC’s new PC1A package C-state as soon as all cores are in CC1. Fig. 6(a) shows core C-state residency for the $C_{\text{shallow}}$ baseline; the average fraction of time each core is in CC0...
and CC1 core C-states. For low load (<100K QPS), we observe that for a large fraction of time (at least 76% to 98%) a core is in CC1. Entering PC1A, however, requires all cores to concurrently be present at CC1; Fig. 6(b) quantifies this opportunity. Since the baseline system, we use to emulate APC, does not actually implement the PC1A state, we estimate PC1A residency as the fraction of time when the system is fully idle, i.e., all cores are simultaneously in CC1. We collect this information through SoCwatch, as described in Sec. 6. We observe that, although PC1A residency diminishes at high load, the opportunity is significant (≥12%) at low load (<100 QPS), with PC1A residency reaching 77% at 4K QPS and 20% for 50k QPS. Fig. 6(c) provides further details on the distribution of the length of fully idle periods (i.e., all cores in CC1). We observe that, at low load, 60% of the idle periods have a duration between 20µs and 200µs, whereas the PC1A transition latency is ≤200µs. The fast PC1A transition latency enables to reap most of the power reduction opportunity during short periods with all cores idle. This is infeasible with existing PC6 state, which has almost no power saving opportunity with its >50µs transition latency.

Since servers running latency-critical applications typically operate at low load, we conclude that real deployments have significant opportunity to enter APC’s new PC1A C-state and benefit from its power savings, which we discuss next.

7.2 PC1A Power Savings

Having confirmed the opportunity to enter package C-state PC1A, we now study the power savings we can expect from APC. Fig. 7(a) shows the processor SoC and DRAM power consumption when all cores are idle for three different configurations: $C_{shallow}$ baseline, $C_{deep}$ baseline, and $C_{PC1A}$. $C_{PC1A}$ corresponds to the $C_{shallow}$ configuration enhanced with our new PC1A package C-state. We estimate idle package power and idle DRAM power of $C_{PC1A}$ using our power analysis discussed in Sec. 5. Idle power for the $C_{PC1A}$ configuration is at a middle point between the $C_{shallow}$ (i.e., no package power savings) and the $C_{deep}$ (i.e., deep C-states enabled, but unrealistic for servers). More specifically, $C_{PC1A}$ enables 41% lower idle power consumption than the $C_{shallow}$.

Fig. 7(b) reports 1) the $C_{shallow}$ baseline and $C_{PC1A}$ power consumption, and 2) $C_{PC1A}$’s power savings as compared to the $C_{shallow}$ baseline for varying request rates (QPS). We observe that $C_{PC1A}$ has lower (or equal) power consumption than the baseline system across the entire range of request rates. The power savings are more pronounced at low load, where the opportunity to enter the PC1A state is higher, as discussed in Sec. 7.1. At 4k QPS, the $C_{PC1A}$ configuration has 37% lower power, while at 50K QPS, it has 14% lower power. The 0K QPS represents the expected power savings during idle periods, when no tasks are assigned to the server.

We conclude that the new deep package C-state, PC1A, results in significant power savings during fully idle periods and at low load, the operating points in which modern server have poor energy efficiency [62], thus making the datacenter servers more energy proportional.

7.3 PC1A Performance Impact

Although PC1A makes the system more energy proportional, entering and exiting PC1A introduces a small (<200µs) transition overhead. Fig. 7(c) analyzes the impact of APC on average end-to-end latency for different request rates, according to our methodology described in Sec. 6. End-to-end latency includes server-side latency plus network latency, which accounts to ≈117µs.

To estimate the performance degradation for different request rates, our performance model uses 1) the number of PC1A transitions, 2) the distribution of number of active cores after exiting full idle, and 3) the transition cost (200ns). We observe that even in the worst case, PC1A has a negligible impact (<0.1%) on average latency. While we do not show additional results due to space constraints, we observe that the overhead on end-to-end tail latency is even smaller.

We conclude that PC1A is a practical package C-state that improves energy proportionality for datacenter servers with negligible performance degradation.

7.4 Analysis of Additional Workloads

![Figure 8: Evaluation of MySQL for low, mid and high request rates. (a) Residency of the $C_{shallow}$ baseline and $C_{PC1A}$ at different core C-states and PC1A. (b) Average power reduction of the $C_{PC1A}$ configuration as compared to the $C_{shallow}$.](image)
Previous work proposes fine-grained DVFS control to save power, with avoiding excessive latency degradation. Rubik [52] scales core frequency at sub-ms scale based on a statistical performance model to save power, while still meeting target tail latency requirements. Swan [90] extends this idea to computational sprinting (e.g., Intel Turbo Boost): requests are initially served on a core operating at low frequency and, depending on the load, Swan scales the frequency up (including sprinting levels) to catch up and meet latency requirements. NMAP [50], focuses on the network stack and leverages transitions between polling and interrupt mode as a signal to drive DVFS management. The new PC1A state of APC facilitates the effective use of idle states and makes a simple race-to-halt approach more attractive compared to 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 decisions. SleepScale [61] is a runtime power management tool that selects the most efficient C-state and DVFS setting for a given QoS constraint based on workload profiling information. WASP [88] proposes a two-level power management framework; the first level tries to steer bursty request streams to a subset of servers, such that other machines can leverage deeper, longer-latency idle states; the second level adjusts local power management decisions based on workload characteristics such as job size, arrival pattern and system utilization. Similarly, CARB [89] tries to pack requests into a small 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 onto a subset of active cores, so as to extend quiet periods on other cores is further explored by other work focusing on both C-state and DVFS management [7, 16, 17]. These proposals are orthogonal to APC and can bring additive improvements. In particular, a technique that synchronizes active / idle periods across different cores while curbing latency degradation can increase the duration of system-level idle periods and, subsequently, the power-saving opportunity.

9. CONCLUSION

This paper presents the design of AgilePkgC (APC): a new C-state architecture that improves the energy proportionality of servers that operate at low utilization while running microservices of user-facing applications. APC targets the reduction of power when all cores are idle in a shallow C-state ready to transition back to service. In particular, APC targets the power of the resources shared by the cores (e.g., LLC, network-on-chip, IOs, DRAM) which remain active while no core is active to use them. APC realizes its objective by using low-overhead hardware to facilitate sub-microsecond entry/exit latency to a new package C-state and judiciously selecting intermediate power modes, for the different shared resources, that offer fast transition and, yet, substantial power savings. Our experimental evaluation supports that APC holds potential to reduce server power of up to 41% with a worst case performance degradation less than 0.1% for several representative workloads. Our results clearly support for the research and development and eventual adoption of new deep and fast package C-states, likes APC, for future server CPUs targeting datacenters running microservices.

Acknowledgments

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 101029391.
References

[1] “A 22nm 2.5 MB Slice on-die L3 Cache for the Next Generation Xeon® processor.”

[2] “Memcached: A distributed memory object caching system,” online, accessed April 2022 https://memcached.org/.

[3] “Powertop,” online, accessed April 2022 https://github.com/fenrus75/powertop.

[4] “sysbench: Scriptable database and system performance benchmark,” online, accessed April 2022 https://github.com/akopytov/sysbench.

[5] F. Alazemi, A. Azizimazreah, B. Bose, and L. Chen, “Routerless Network-on-Chip,” in HPCA, 2018.

[6] R. Appuswamy, M. Olma, and A. Ailamaki, “Scaling the memory power wall with dram-aware data management,” in DaMoN, 2015.

[7] E. Asyabi, A. Bestavros, E. Sharafzadeh, and T. Zhu, “Peafowl: In-application CPU Scheduling to Reduce Power Consumption of In-memory Key-value Stores,” in SoCC, 2020.

[8] B. Atikoglu, Y. Xu, E. Frachtenberg, S. Jiang, and M. Paleczny, “Workload Analysis of a Large-scale Key-value Store,” in SIGMETRICS, 2012.

[9] L. Barroso, M. Marty, D. Patterson, and P. Ranganathan, “Attack of the killer microseconds,” Communications of the ACM, 2017.

[10] L. A. Barroso, U. Hölzle, and P. Ranganathan, “The datacenter as a computer: Designing warehouse-scale machines,” Synthesis Lectures on Computer Architecture, vol. 13, no. 3, pp. 1–189, 2018.

[11] R. Budruk, D. Anderson, and T. Shanley. PCI express system architecture. Addison-Wesley Professional, 2004.

[12] E. A. Burton, G. Schrom, F. Paillet, J. Douglas, W. J. Lambert, K. Radhakrishnan, and M. J. Hill, “FIVR - Fully integrated voltage regulators on 4th generation Intel® Core™ SoCs,” in APEC, 2014.

[13] K. Chandana and R. Karunavathi, “Link Initialization and Training in MAC Layer of PCIe 3.0,” IJCSIT, 2015.

[14] W. Chen, S.-L. Chen, S. Chiu, R. Ganesan, V. Lukka, W. W. Mar, and S. Rusu, “Presentation of: A 22nm 2.5MB slice on-die L3 cache for the next generation Xeon® Processor,” 2013, https://bit.ly/3bYXDJe.

[15] S. Cho, A. Suresh, T. Palit, M. Ferdman, and N. Honarmand, “Taming the Killer Microsecond,” in MICRO, 2018.

[16] C.-H. Chou, D. Wong, and L. N. Bhuyan, “Dynsleep: Fine-grained Power Management for a Latency-critical Data Center Application,” in ISLPED, 2016.

[17] C.-H. Chou, L. N. Bhuyan, and D. Wong, “μDPM: Dynamic Power Management for the Microsecond Era,” in HPCA, 2019.

[18] CPUbenchmark, “AMD vs Intel Market Share,” accessed Nov 2020, https://bit.ly/3kV6kWY.

[19] H. David, C. Fallin, E. Gorbatov, U. R. Hanebutte, and O. Mutlu, “Memory Power Management via Dynamic Voltage/Frequency Scaling,” in ICAC, 2011.

[20] Dell, “PowerEdge: DRAM Refresh delay and Opportunistic Self-Refresh,” online, accessed March 2022, Mar, https://bit.ly/37frHSi.

[21] N. Dmitry and S.-S. Manfred, “On Micro-Services Architecture,” INJOIT, vol. 2, no. 9, 2014.

[22] A. M. El-Husseini and M. Morrise, “Clocking Design Automation in Intel’s Core 17 and Future Designs,” in ICCAD, 2011.

[23] M. Fahad, A. Shahid, R. R. Manumach, and A. Lastovetsky, “A Comparative Study of Methods for Measurement of Energy of Computing,” Energies, 2019.

[24] C. Fallin, X. Yu, G. Nazario, and O. Mutlu, “A High-performance Hierarchical Ring on-chip Interconnect with Low-cost Routers,” Computer Architecture Lab, Carnegie Mellon Univ, Tech. Rep, 2011.

[25] E. Fayneh, M. Yuffe, E. Knoll, M. Zelikson, M. Abozaed, Y. Talker, Z. Shmuely, and S. A. Rahme, “4.1 14nm 6th-generation Core Processor SoC with Low Power Consumption and Improved Performance,” in ISSCC, 2016.

[26] C. Gough, I. Steiner, and W. Saunders, “CPU Power Management,” in Energy Efficient Servers: Blueprints for Data Center Optimization, 2015.

[27] D. Hackenberg, R. Schöne, T. Ilsche, D. Molka, J. Schuchart, and R. Geyer, “An Energy Efficiency feature survey of the Intel Haswell processor,” in 2015 IEEE international parallel and distributed processing symposium workshop. IEEE, 2015, pp. 896–904.

[28] J. Haj-Yahya, A. Mendelson, Y. B. Asher, and A. Chattopadhyay, “Power Management of Modern Processors,” in EEHPC, 2018.

[29] J. Haj-Yahya, E. Rotem, A. Mendelson, and A. Chattopadhyay, “A Comprehensive Evaluation of Power Delivery Schemes for Modern Microprocessors,” in IQED, 2019.

[30] J. Haj-Yahya, E. Rotem, A. Mendelson, A. Chattopadhyay, and O. Mutlu, “FlexWatts: A Power-and Workload-Aware Hybrid Power Delivery Network for Energy-Efficient Microprocessors,” in MICRO, 2020.

[31] J. Haj-Yahya, Y. Sazeides, M. Alser, E. Rotem, and O. Mutlu, “Techniques for reducing the connected-standby energy consumption of mobile devices,” in HPCA, 2020.
[65] D. Meisner, B. T. Gold, and T. F. Wenisch, “Powernap: Eliminating Server Idle Power,” ASPLOS, 2009.

[66] J. Moreira and H. Werkmann, An Engineer’s Guide to Automated Testing of High-Speed Interfaces. Artech House, 2010.

[67] A. Nalamalpu, N. Kurd, A. Deval, C. Mozak, J. Douglas, A. Khanna, F. Paillet, G. Schrom, and B. Phelps, “Broadwell: A family of IA 14nm processors,” in VLSI Circuits (VLSI Circuits), 2015 Symposium on. IEEE, 2015, pp. C314–C315.

[68] R. Nishtala, H. Fugal, S. Grimm, M. Kwiatkowski, H. Lee, H. C. Li, R. McElroy, M. Paleczny, D. Peek, P. Saab, D. Stafford, T. Tung, and V. Venkatararmani, “Scaling Memcache at Facebook,” in 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13), 2013.

[69] D.-V. One, “Intel® xeon® processor e7-8800/4800/2800 v2 product family,” https://intel.ly/2ZGA9FJ.

[70] Oracle, “MySQL Workbench,” online, accessed April 2022 https://www.mysql.com/products/workbench/.

[71] D. Peterson and O. Bringmann, “Fully-automated Synthesis of Power Management controllers from UPF,” in ASP-DAC, 2019.

[72] Pinterest, “pymemcache: A comprehensive, fast, pure-Python memcached client,” online, accessed November 2021 https://github.com/pinterest/pymemcache.

[73] G. Prekas, M. Kogias, and E. Bugnion, “Zygos: Achieving low tail latency for microsecond-scale networked tasks,” in SDP, 2017.

[74] K. Radhakrishnan, M. Swaminathan, and B. K. Bhattacharyya, “Power Delivery for High-Performance Microprocessors—Challenges, Solutions, and Future Trends,” IEEE Transactions on Components, Packaging and Manufacturing Technology, 2021.

[75] E. Rotem, A. Naveh, D. Rajwan, A. Ananthakrishnan, and E. Weissmann, “Power Management Architecture of the 2nd Generation Intel® Core Microarchitecture, Formerly Codenamed Sandy Bridge,” in HotChips, 2011.

[76] E. Rotem, A. Naveh, A. Ananthakrishnan, E. Weissmann, and D. Rajwan, “Power-management architecture of the intel microarchitecture code-named sandy bridge,” IEEE MICRO, vol. 32, no. 2, pp. 20–27, 2012.

[77] E. Rotem, Y. Mandelblat, V. Basin, E. Weissmann, A. Gikon, R. Chabukswar, R. Fenger, and M. Gupta, “Alder Lake Architecture,” in HCS, 2021.

[78] R. Schöne, D. Molk, and M. Werner, “Wake-up Latencies for Processor Idle States on Current x86 Processors,” Computer Science-Research and Development, 2015.

[79] G. Shamanna, N. Kurd, J. Douglas, and M. Morrise, “Scalable, sub-1W, sub-10ps clock skew, global clock distribution architecture for Intel® Core™ i7/i5/i3 microprocessors,” in Symposium on VLSI Circuits, 2010.

[80] stackoverflow, “I can’t see perf’s power/energy-cores option for measure power consumption,” accessed April 2022, https://bit.ly/3JqQHmW.

[81] ——, “Why i can’t use perf event modifiers with power/energy-cores,” accessed April 2022, https://bit.ly/37nMlzB.

[82] S. M. Tam, H. Muljono, M. Huang, S. Iyer, K. Royneogi, N. Satti, R. Qureshi, W. Chen, T. Wang, H. Hsieh, S. Vora, and E. Wang, “SkyLake-SP: A 14nm 28-Core Xeon® Processor,” in ISSCC, 2018.

[83] Wikichip, “Skylake (server) - Microarchitectures - Intel,” online, accessed November 2021 https://bit.ly/2MHEWkj.

[84] ——, “M1 - Apple,” online, accessed April 2022 https://en.wikichip.org/wiki/apple/mx/m1.

[85] ——, “Mesh Interconnect Architecture - Intel,” online, accessed November 2021 https://en.wikichip.org/wiki/intel/mesh_interconnect_architecture.

[86] Y. Xu, E. Frachtenberg, S. Jiang, and M. Paleczny, “characterizing facebook’s memcached workload,” IEEE Internet Computing.

[87] J. Yang, Y. Yue, and K. V. Rashmi, “A large scale analysis of hundreds of in-memory cache clusters at twitter,” in 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20), 2020.

[88] F. Yao, J. Wu, S. Subramaniam, and G. Venkatararmani, “WASP: Workload Adaptive Energy-latency Optimization in Server Farms Using Server Low-power States,” in CLOUD, 2017.

[89] X. Zhan, R. Azimi, S. Kanev, D. Brooks, and S. Reda, “CARB: A C-state Power Management Arbiter for Latency-critical Workloads,” IEEE Computer Architecture Letters, vol. 16, no. 1, pp. 6–9, 2016.

[90] L. Zhou, L. N. Bhuyan, and K. Ramakrishnan, “Swan: a two-step power management for distributed search engines,” in ISLPED, 2020.

[91] Barroso, Luiz André, Jeffrey Dean, and Urs Holzle. “Web search for a planet: The Google cluster architecture.” IEEE micro 2003.

[92] Jeon, M., He, Y., Elnikety, S., Cox, A.L. and Rixner, S., 2013, April. “Adaptive parallelism for web search”. EuroSys 2013.

[93] D. Wong and M. Annavaram, “Knightshift: Scaling the Energy Proportionality Wall Through Server-level Heterogeneity”. In MICRO, 2012.
[94] S. Luo, H. Xu, C. Lu, K. Ye, G. Xu, L. Zhang, Y. Ding, J. He, and C. Xu. “Characterizing Microservice Dependency and Performance: Alibaba Trace Analysis”. In SoCC, 2021.

[95] Github, “Spark-Bench”

[96] Apache, “Apache Hive”