

Article

Performance Characterization of Hardware/Software Communication Interfaces in End-to-End Power Management Solutions of High-Performance Computing Processors

Antonio del Vecchio ^{1,*} , Alessandro Ottaviano ² , Giovanni Bambini ¹ , Andrea Acquaviva ¹ 
and Andrea Bartolini ^{1,*} 

¹ Department of Electrical, Electronic, and Information Engineering, University of Bologna, 40126 Bologna, Italy; giovanni.bambini2@unibo.it (G.B.); andrea.acquaviva@unibo.it (A.A.)

² Integrated Systems Laboratory, ETH Zurich, 8092 Zurich, Switzerland

* Correspondence: antonio.delvecchio5@unibo.it (A.d.V.); a.bartolini@unibo.it (A.B.)

Abstract: Power management (PM) is cumbersome for today's computing systems. Attainable performance is bounded by the architecture's computing efficiency and capped in temperature, current, and power. PM is composed of multiple interacting layers. High-level controllers (HLCs) involve application-level policies, operating system agents (OSPMs), and PM governors and interfaces. The application of high-level control decisions is currently delegated to an on-chip power management unit executing tailored PM firmware routines. The complexity of this structure arises from the scale of the interaction, which pervades the whole system architecture. This paper aims to characterize the cost of the communication backbone between high-level OSPM agents and the on-chip power management unit (PMU) in high performance computing (HPC) processors. For this purpose, we target the System Control and Management Interface (SCMI), which is an open standard proposed by Arm. We enhance a fully open-source, end-to-end FPGA-based HW/SW framework to simulate the interaction between a HLC, a HPC system, and a PMU. This includes the application-level PM policies, the drivers of the operating system-directed configuration and power management (OSPM) governor, and the hardware and firmware of the PMU, allowing us to evaluate the impact of the communication backbone on the overall control scheme. With this framework, we first conduct an in-depth latency study of the communication interface across the whole PM hardware (HW) and software (SW) stack. Finally, we studied the impact of latency in terms of the quality of the end-to-end control, showing that the SCMI protocol can sustain reactive power management policies.

Keywords: RISC-V; Power Management; DVFS; Arm SCMI; Hardware in the Loop; HPC



Citation: del Vecchio, A.; Ottaviano, A.; Bambini, G.; Acquaviva, A.; Bartolini, A. Performance Characterization of Hardware/Software Communication Interfaces in End-to-End Power Management Solutions of High-Performance Computing Processors. *Energies* **2024**, *17*, 5778. <https://doi.org/10.3390/en17225778>

Academic Editor: Wiseman Yair

Received: 20 September 2024

Revised: 30 October 2024

Accepted: 6 November 2024

Published: 19 November 2024



Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

The growing computational demand for artificial intelligence, big data, and scientific computing applications has led to a significant rise in energy consumption for both data centers and HPC systems. Recent studies show that energy consumption in data centers has been steadily increasing, reflecting a broader trend also seen in HPC environments [1]. In these systems, energy is typically split between computational elements and infrastructure overhead, including cooling and power distribution. Cooling alone accounts for 24% to 60% of total energy consumption, depending on the system architecture and operational workload [1]. Of the remaining energy, approximately 30% is consumed by computing elements, with the remainder consumed by memory, interconnects, and power losses due to conversion inefficiency [2]. In this context, the power management of computing elements is crucial, providing the opportunity to optimize the power consumption of more than one-third of the total energy demand as cooling power decreases with computing power demand.

Until two decades ago, PM was an exclusive responsibility of the Operating System (OS) running on top of application-class processors (AP). The agent handling PM in the OS is known as OSPM [3]. Placing all the power policy responsibility on the OSPM brings several drawbacks. Firstly, the complex interaction between power, temperature, workload, and physical parameters in an integrated system on chip (SoC), coupled with additional safety and security requirements, might be too complex for the OS to manage while simultaneously optimizing workload performance [4]. Secondly, the OS does not have introspection on the application behavior and events, but can only operate on timer-based tick instants, or slower. Finally, in a central processing unit (CPU) thermal constant, power and current can vary so quickly that OSPM SW is unable to deal with them timely [3].

Historically, these issues called for a paradigm shift in PM responsibilities within an integrated system. The new paradigm transitions from an OS-centric to a delegation-based model where APs (or HLCs) collaborate with general-purpose, embedded low-level controllers (LLCs). The implications of such a model were not fully taken into account when industry-standard PM firmware (FW), such as Advanced Configuration and Power Interface (ACPI) described in Section 2.4.1, was first proposed. Throughout the years, HPC and mobile platforms from leading industry competitors started adopting this model with proprietary HW/SW architectures [5].

A delegation-based scheme implies clear responsibility boundaries between the HLC and the LLC. Therefore, a key design element becomes the power management interface (PMI) between these two components. As the backbone of the connection traversing two (possibly) heterogeneous SW and HW stacks, its design demands several features: (i) OS and PM FW independence, (ii) modularity, (iii) flexibility, and (iv) low-latency overhead. The last property is particularly important in a delegation-based approach: the LLC is subject to soft real-time constraints and periodically applies the power policy.

To date, power management evaluations have mainly focused on optimizing dynamic voltage and frequency scaling (DVFS) policies according to application performance patterns and the underlying architectural power profiles [6,7]. Additionally, some efforts have concentrated on control algorithms to dynamically adjust voltage and frequency operating points within multicore systems [8]. In HPC systems, where dynamic workloads are prevalent, the delay between a change in workload and the application of optimal operating points for each core can degrade end-to-end control quality, leading to reduced energy efficiency. Therefore, an exhaustive analysis of latency overhead introduced by PMIs is essential to fully understand how it affects the performance of modern HPC processors.

In this work, we carry out this exploration by taking the example of Arm-based server systems and their open-standard SCMI protocol. As of today, SCMI is the only OSPM standard with a holistic and open description of PMIs, seamlessly coupled with existing industry-standard FW, e.g., ACPI. We justify our choice in detail in Section 2.4.2.

To conduct this assessment, we employ an open-source, HW/SW LLC design, coupled with a field-programmable gate array (FPGA) implementation of a hardware in the loop (HIL) framework for power and thermal management simulation [5]. Such a framework is essential for the type of analysis conducted, as it enables the emulation of the key elements constituting the power management scheme of a processor, allowing for the inspection of all interactions occurring among these components. It relies on (i) an Arm-based HW platform co-design, and (ii) a Xilinx Ultrascale+ FPGA leveraging a fully-featured, SCMI-capable Linux SoC, as detailed in Section 3.

To the best of the authors' knowledge, this is the first work providing a fine-grained and quantitative insight into modern PMI behavior, and their impact on runtime PM. The framework is released as open-source.

Contribution

We present the following contributions:

- We extend the FPGA-based HIL introduced in [5] to analyze the communication backbone between high-level OSPM agents to the LLC's HW and FW by integrating

the Linux SCMI SW stack and by implementing an SCMI mailbox unit (SCMI-MU) compliant with MHU-v1 functionality.

- We characterize the performance of HW/SW low-level PMIs in Arm architectures based on latency metrics. We quantify the duration for dispatching an SCMI message through the Linux OSPM stack, resulting in a time of 70.5 μ s. Additionally, we measure the processing time of a SCMI response message, yielding an average of 603 μ s.
- Using the developed setup, we focus on the entire power management control scheme, assessing how different configurations—namely a periodic and event-driven HLC—introduce latency in the PMI and, consequently, how it affects the end-to-end power management control quality, showing that an acceleration of up to 2.7% can be achieved in the execution time of a synthetic workload. Ultimately, through the development of an optimized version of the LLC's control FW, we demonstrate that the latency introduced by the PMI can be reduced from around 1.3 ms to 114 μ s, thereby reducing the energy consumption and application execution time of approximately 3%.

2. Background and Related Works

2.1. Overview and Terminology

In this section, we recall the main terminology and HW/SW components for PM of modern HPC SoCs, pictured in Figure 1. The taxonomy is based on the three largest domains—HLC, LLC, and their interface.

The UEFI's PM ACPI specification distinguishes among three types of PM: *system*, *device*, and *processor* [9]. Similarly, Arm differentiates between *system*, *device*, and *core* PM services [10]. *Processor* (or *core*) and *device* services are typically requested by the OSPM, while *system* services are provided without OSPM mediation, for example, by the Baseboard Management Controller (BMC) on the motherboard [10]. This work focuses on the OSPM-directed *processor* and *device* PM. We will refer to a single general-purpose compute unit of a many-core system as processing element (PE).

The LLC periodically interfaces with Process, Voltage, Temperature (PVT) sensors and actuators and responds to HLC's directives from the OSPM and user's applications through dedicated HW and SW PMIs, discussed in Section 2.4. These on-die interactions are collectively known as *in-band* services [5,11], and are the main focus of the present analysis. Finally, the LLC interfaces with the BMC on the motherboard to support off-chip *system* services, also called *out-of-band* [11]. These comprise fine-grain telemetry on the chip power and performance status, chip-level and system-level power capping, and reporting errors and faults in the chip and central processes.

2.2. HLC Components

2.2.1. HW Layer

Modern server-class SoCs are heterogeneous many-core chiplet architectures. Each chiplet integrates tens to hundreds of AP PEs, graphic processing units (GPUs), and domain-specific accelerators (DSAs). Chiplets communicate through performant links and interface with advanced memory endpoints, such as 3D high-bandwidth memory (HBM) stacks.

2.2.2. PM SW Stack

The HLC's OSPM is the agent that controls the system's power policy [3]. We rely on terminology adopted in the Linux OSPM, used in this work. The usual tasks subsumed by the OSPM are APs idle and performance PM, device PM, and power monitoring. These tasks are managed by OSPM *governors*, routines tightly coupled with the OS' kernel scheduling policy. An HPC workload consists of parallel applications distributed across all PEs of a set of processors, with one process per core. Under these conditions, the performance, powersave, and userspace governors exhibit the same behavior—selecting the highest operating frequency. However, state-of-the-art processor PM strategies often leverage application PM runtime, which isolates different phases of the application and requests a new DVFS level from the OSPM at phase transitions. Two major approaches exist

for identifying application phases, (i) *timer-based* (or *periodic*), which involves periodically reading performance counters to identify computational bottleneck regions (i.e., memory- or CPU-bound), and (ii) *event-driven*, which uses application code instrumentation (which can be programmer-driven, parallel programming-driven, or compiler-driven) to flag the entry into different code regions.

2.3. LLC Components

2.3.1. HW Layer

LLCs are usually 32-bit microcontrollers with optional general-purpose or domain-specific modules, from efficient data moving engines to microcode-driven co-processors or programmable many-core accelerators (PMCA) to accelerate the PM policy. The LLC is subject to soft and hard real-time requirements, thus demanding streamlined interrupt processing and context switch capabilities [12]. Moreover, its I/O interface has to sustain *out-of-band* communication through standard HW PMIs, such as Power Management Bus (PMBUS) and Adaptive Voltage Scaling Bus (AVSBus).

2.3.2. PM SW Stack

The PM policy can be scheduled as a bare-metal FW layer [13,14], or leverage a lightweight real-time OS (RTOS) (e.g., FreeRTOS), as in the open-source ControlPULP [5] employed in this work. The latter is based on RISC-V parallel cores and an associated cascade control PM FW. It supports the co-simulation of the design with power and thermal simulators in Xilinx Zynq Ultrascale+ ZCU102 FPGA, as reported in Figure 1b.

2.4. HLC-LLC Interface

2.4.1. Industry-Standard FW Layer

ACPI [9] is an open standard framework that establishes a HW register set (*tables* and *definition blocks*) to define power states. The primary intention is to enable PM and system configuration without directly calling FW natively from the OS. ACPI provides a set of PM services to a compliant OSPM implementation: (1) Low Power Idle (LPI) to handle power and clock gate regulators; (2) device states; (3) collaborative processor performance control (CPPC) for DVFS enforcement; and (4) power meters for capping limits. CPPC allows the OSPM to express DVFS performance requirements to the LLC, whose FW makes a final decision on the selected frequency and voltage based on all constraints. The communication channel between HLC and LLC is formally defined as platform communication channel (PCC) by ACPI; it can be a generic mailbox mechanism, or rely on fixed functional hardware (FFH), i.e., ACPI registers hardwired for specific use.

2.4.2. OS-Agnostic FW Layer

Most industry vendors rely on proprietary interfaces between HLC and LLC. Intel uses model-specific registers (MSRs) mapped as ACPI's FFH to tune PEs' performance. Performance requests are handled through the `intel_pstate` governor in Linux, while capping is enforced through the Running Average Power Limit (RAPL) framework. In IBM systems, the OpenPower abstraction layer (OPAL) framework relies on special purpose registers (SPRs) and a shared memory region in the On-Chip Controller (OCC) to interact with the LLC. In the RISC-V ecosystem, recent efforts in designing HPC systems have led to specific ACPI extensions and PMIs, known as RPMI (<https://github.com/riscv-non-isa/riscv-rpmi>, accessed on: 20 September 2024).

Arm, on the other hand, avoids the limitations imposed by FFHs and proposes a more flexible SCMI protocol to handle HLC to LLC performance, monitoring, and low-power regulation requests. SCMI involves an interface channel for secure and non-secure communication between an *agent*, e.g., a PE, and a *platform*, i.e., the LLC. The platform receives and interprets the messages in a shared memory area (mailbox unit) and responds according to a specific protocol. The design of the SCMI protocol reflects the industry trend of delegating power and performance to a dedicated subsystem [4] and provides a

flexible abstraction that is platform-agnostic. In this manuscript, we evaluate the cost of this delegation in terms of PM performance. We first characterize the inherent delay of traversing HW/SW structures, and its impact in end-to-end (HLC-to-LLC) PM. Then, we propose an optimization for the FW PM, which leverages the SCMI associated latency. The next section introduces the methodology proposed for conducting this co-design.

3. Methodology: HIL Framework on FPGA

Figure 1c shows the block diagram of the proposed HIL emulation framework to model and evaluate the PMI in the end-to-end PM HW/SW stack. The lower layer comprises the underlying ControlPULP controller, i.e., its register transfer level (RTL) HW description (1). ControlPULP is emulated in the Programmable Logic (PL) of the FPGA-SoC, to which we added a SCMI-MU (2) to provide the HW transport for SCMI protocol. At the same time, the Linux OS image (3) running on the Processing System (PS) has been modified to propagate OSPM requests to the ControlPULP LLC via the SCMI-MU. Additionally, a shared memory interface (4) is in place to emulate PM virtual sensors and actuators. Indeed, the simulated plant (5) provides a thermal, power, performance, and monitoring framework to simulate the power consumption and temperature of a high-end CPU. The plant simulation is programmed in C and runs on the PS's Arm A53 cores.

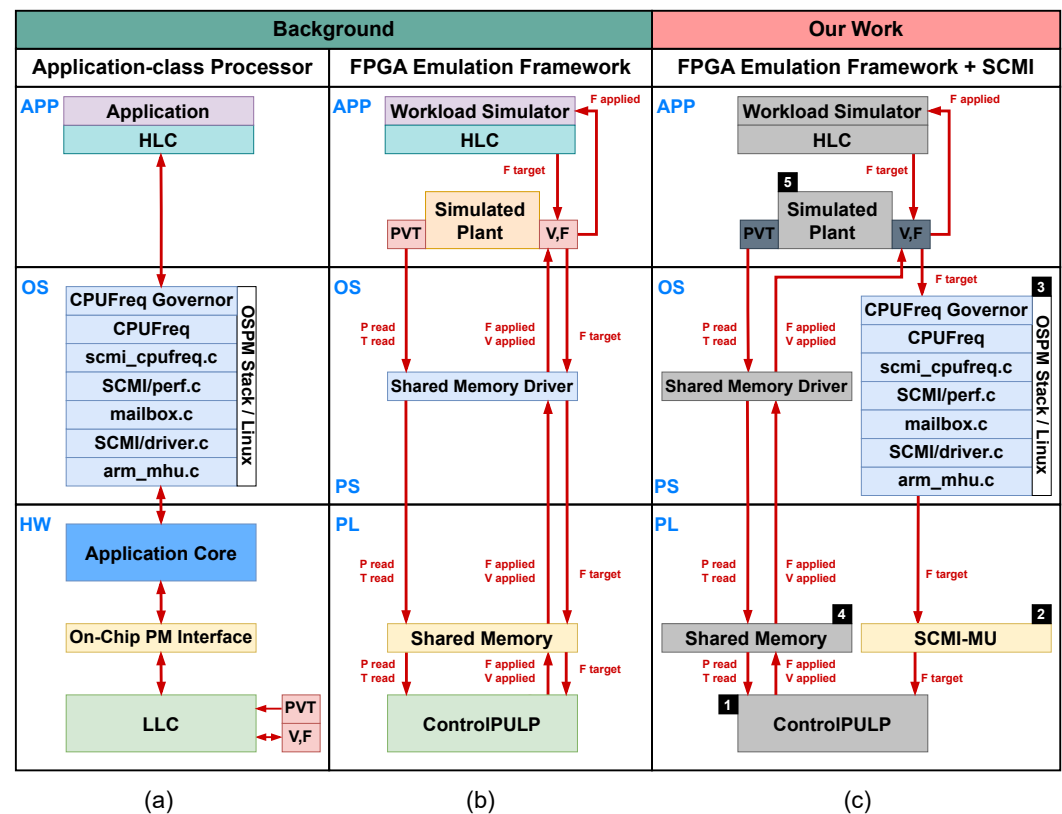


Figure 1. Overview of the main PM components of modern HPC SoCs. This work focuses on the PMIs linking HLCs to the LLC from a HW and SW perspective in Arm-based server-class CPUs. (a) Architecture of the power management scheme in an application-class processor, (b) Block diagram of a state-of-the-art HIL platform for power management systems emulation, (c) Block diagram of the extended HIL platform proposed in this work.

3.1. SCMI-MU Implementation

We implement a HW mailbox unit, named SCMI-MU, according to Arm's SCMI standard, designed to be compatible with high-level SCMI drivers in the Linux kernel. Its design provides all the MHU-v1 functionality [15], ensuring alignment with the corresponding low-level drivers in the Linux system. As shown in Figure 1, the SCMI-MU is accessible from both the PS and the LLC through a 32-bit AXI Lite front-end. The SCMI-MU implements 33 32-bit

registers for SCMI compliance and serves as a shared space for storing host messages from multiple agents. The interrupt generation logic comprises two 32-bit registers, named doorbell and completion, respectively [4], and an interrupt generation logic. Both registers generate a level-sensitive, HW interrupt to the LLC/HLC when set and gate the interrupt when cleared. On the LLC side, the doorbell interrupts are routed through a RISC-V Core-Local Interrupt Controller (CLIC), which handles ControlPULP hardware interrupts and improves real-time performances [12]. On the HLC side, the completion interrupts are routed through the Arm GIC-V2. When a message is fetched by the LLC, the doorbell register is cleared so as not to block subsequent writes. This mechanism lets the sender know if the receiver has fetched the message.

3.2. Linux SCMI SW Stack

As mentioned in Section 2.2, in Linux, application-level PM policies require the `cpufreq_userspace` governor to set frequency setpoints. The SCMI stack is responsible for the interaction between the governor and the LLC and is composed of the following main routines: `cpufreq`, `scmi_cpufreq`, `perf`, `mailbox`, `arm_scmi` driver, and `arm_mhu`. The HLC writes a target frequency value for core i in an interface file in the system virtual filesystem in Linux which automatically calls the `cpufreq` driver methods. We use release v4.19.0 of the Linux kernel, compatible with Arm MHU-v1, where the `cpufreq` and `scmi_cpufreq` are tightly coupled for message transmission and reception. The trellis graph in Figure 2 shows the interaction between the Linux kernel drivers upon the arrival of a new target frequency from the user.

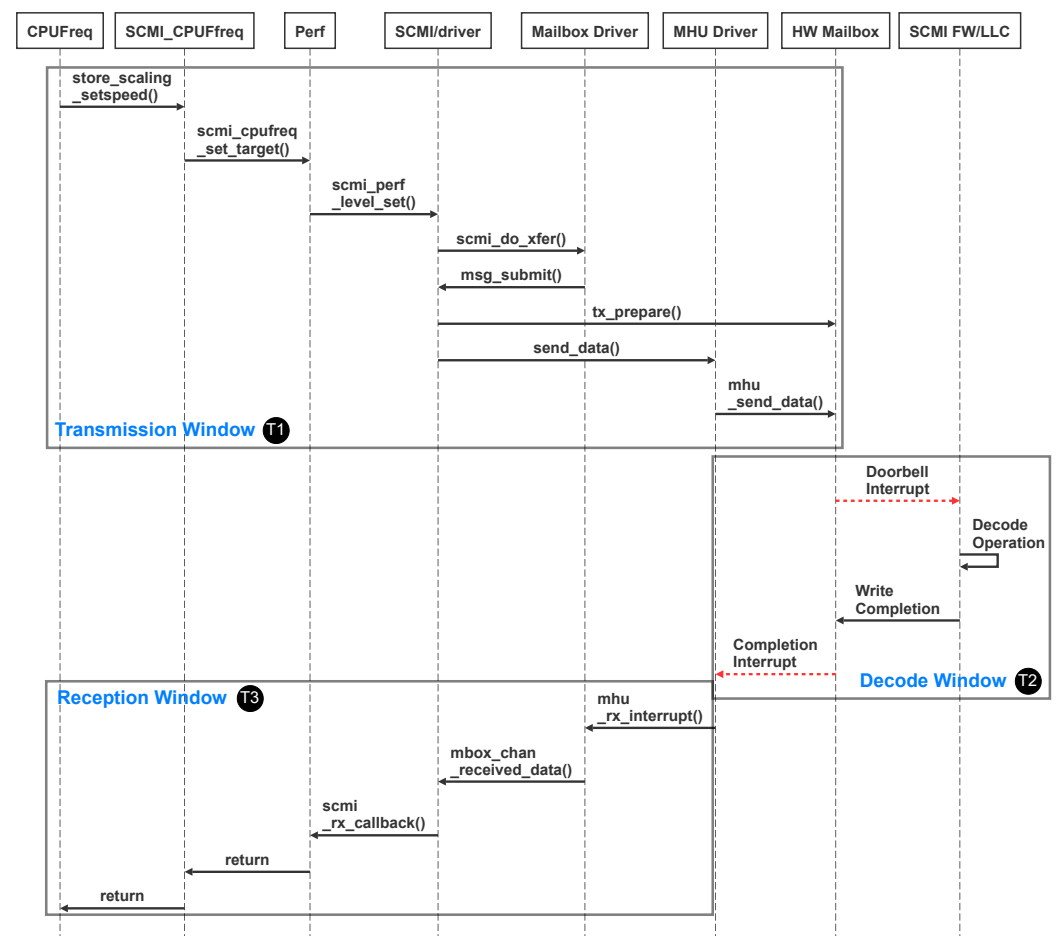


Figure 2. Sequence of function calls within the Linux CPUFreq stack, for a `perf_level_set` SCMI request.

The `scmi_cpufreq` driver directly calls `perf`, which prepares an SCMI message through the `perf_level set` command. The drivers in the bottom layers transmit the message through the SCMI-MU.

Once the message is sent, ControlPULP processes the request, applies the target frequencies to the cores, and triggers the completion signal. On the OSPM, such a signal is mapped to an interrupt service routine (ISR) handled by the `arm_mhu` driver, which reads the transaction identifier, decodes the message in shared memory, and resets the completion register.

Figure 2 is split into three parts: the *transmission window* T1 on the agent side, ranges from `store_scaling_setspeed()` in `cpufreq` to message delivery in the shared memory by `arm_mhu`; the *decode window* T2 on both platform and agent sides ranges from the doorbell-triggered ISR hook in the LLC to the completion-triggered ISR hook in the HLC via `arm_mhu`; and the *reception window* T3 on the agent side covers function calls until the return of `cpufreq`.

3.3. LLC SCMI FW Module

The SCMI FW module executes on the ControlPULP LLC. The primary purpose of this module is to enable the exchange of SCMI messages between the governor, running on the HLC, and the power policy FW executing on the LLC.

Note that SCMI channel management operations are subject to timing constraints imposed by the Linux SCMI drivers through SW timers tasked to detect channel congestion, upon expiration of which the ongoing transaction may be canceled. The native FreeRTOS-based SW stack on top of the LLC is leveraged to ensure fast platform-agent reaction time in compliance with these timing constraints.

The FW comprises two sections: (1) a low-level, SCMI-MU HW management layer and (2) a high-level decoding layer for the governor's command. The management layer contains methods to access the SCMI-MU shared memory, as well as populating its registers related to interrupt generation. The decoding layer, on the other hand, is embedded within ControlPULP's power control firmware (PCF) as a FreeRTOS task, called *decoding task*, which leverages the methods exposed by the hardware management layer to access messages from/to the SCMI-MU. After decoding the values of the *header* and *payload* fields of the message, the FW maps the SCMI message into a command. The SCMI specification supports a wide range of message types; the firmware implemented in this work focuses on *perf_level set* commands to handle new frequency setpoint values for a given performance domain, common to all cores in our simulation setup.

On a new message arrival, a doorbell interrupt signal triggers an ISR, which saves the message waiting for the decoding task to execute. The latter is queued for execution until scheduled by FreeRTOS, having lower priority compared to the power and thermal policy tasks. The decoding task reads the transaction identifier from the doorbell register and the rest of the message from the shared memory. It then clears the doorbell register to signal message reception to the sender and performs decoding. After decoding, it populates the shared memory with the response and rings the completion interrupt.

3.4. LLC PM Policy Optimized for Latency

In this manuscript, we propose an LLC PM policy optimized for latency. The default ControlPULP PM policy (PCF) [5] relies on a periodic control task executed every 500 μ s (configured to account for ms-scale temperature time constant and μ s-scale phase-locked loop (PLL) locking time); it executes a cascade of a model-based power capping algorithm and PID thermal capping algorithm. The SCMI new frequency setting is read by the algorithm during the 500 μ s period to compute the novel power management settings (PLL frequencies and voltage level), which is then applied in the next task period. This induces a latency of at least one period between the receipt of an incoming SCMI message and a change in the output frequency of the control task. This is intrinsic to the control scheme stability and thermal and power capping capabilities. We propose to bypass this

latency and directly apply the new SCMI setting if the new operating point has a lower frequency, and yet power, with regard to the PCF's computed one for the previous cycles. This allows us to reduce the latency in case of HLC requests, which reduces the actual power consumption. We named this policy LLC Optimized.

3.5. HLC Policy

The HLC is a SW component tasked to analyze the executed workload and generate target frequencies f_T^i for each i -th controlled core, communicated to the LLC via SCMI PMI, as in Figure 1. Although relying on actual SCMI drivers and the Linux stack for command transmission, LLC's responses to SCMI commands are still applied to the simulated cores.

In this manuscript, we consider two different HLC policies: a periodic one (HLC_t) and an event-driven one (HLC_e). The HLC_t computes the target frequency for each simulated core f_T^i with a global periodicity of T_{f_T} for each simulation time step k using a last-value predictor. Indicating with β the maximum acceptable execution time overhead within the next T_{f_T} , $f_{T,max}^i$, the maximum frequency, N_{im} , and N_{ic} the number of memory-bound and CPU-bound instructions executed during the previous T_{f_T} , CPI_{ic} and CPI_{im} , the cycle-per-instruction metrics of compute-bound and memory-bound instructions, respectively [16], $f_T^i[k]$ read as follows [17]:

$$f_T^i[k] = \frac{f_{T,max}^i \cdot N_{ic}[k-1] \cdot CPI_{ic}[k-1]}{N_{im}[k-1] \cdot CPI_{im}[k-1] \cdot (\beta - 1) + N_{ic}[k-1] \cdot CPI_{ic}[k-1] \cdot \beta} \quad (1)$$

Differently, the HLC_e assumes perfect phases instrumentation and selects the right f_T using an oracle at phase transition. This HLC policy emulates the best-case scenario for instrumentation-based power management policies.

4. Evaluation

Section 4.1 characterizes the SCMI call stack and evaluates its latency. Section 4.2 evaluates the impact of PMIs on an end-to-end PM policy running on FPGA-HIL.

4.1. SCMI Latency Characterization

The test has been conducted, setting in round-robin three different frequency setpoints to the `cpufreq_userspace` with a 1 s period. This is larger than the SCMI latency, avoiding message collisions. The first latency analysis focuses on the three HLC time windows described in Section 3.1, namely, from calling `store_scaling_setspeed()` to `cpufreq_notify_transition()`. The second focuses on the LLC FW, measuring its latency from the SCMI notification.

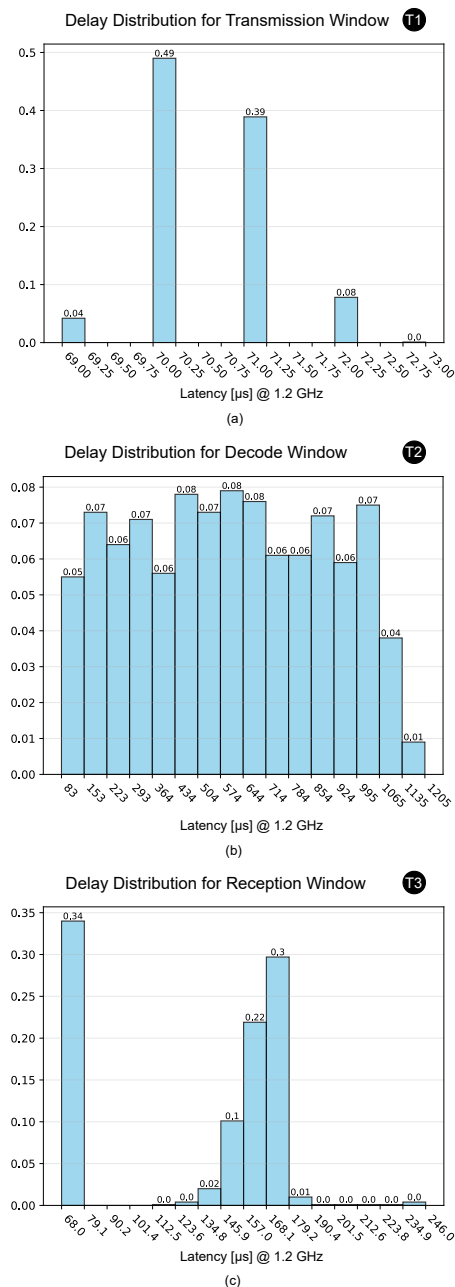
4.1.1. HLC End-to-End Latency Analysis

We use the Linux kernel tool `ftrace` for a fine-grained breakdown of SCMI-related function calls. `ftrace` logs them with a timing precision of 1 μ s. We applied it to 1000 frequency setpoint requests. In all the tests conducted in this paper, the PS was configured to run at 1.2.

The total time for executing a `perf_level set` request is detailed in Table 1a. We show minimum, maximum, and average time across the 1000 setpoint requests sent to the LLC. From Table 1a, we notice that T1 to T3 account for about 8.7%, 74.7%, and 16.5% of the total average time, respectively. Their distribution over 1000 samples is shown in Figure 3. We observe that T1 exhibits a probability distribution falling within a narrow range of only 3 μ s, so we consider its mean value of 70.5 μ s as relevant. T2 shows an almost uniform distribution in the range between 83 μ s and 1205 μ s, while for T3, we can notice a bimodal distribution with two peaks centred around the 75 μ s and 164 μ s.

Table 1. (a) SCMI call stack time measured at 1.2 GHz, (b) ControlPULP SCMI decoding time @20 MHz.

(a)							(b)				
	Minimum [μs]	Minimum [Cycles]	Average [μs]	Average [Cycles]	Maximum [μs]	Maximum [Cycles]		Increment [μs]	Increment [Cycles]	Sum [μs]	Sum [Cycles]
T1	69	82.60 k	70.50	84.60 k	73	87.60 k	CLIC to ISR	2.30	46	2.30	46
T2	83	99.60 k	603.50	724.20 k	1205	1446 k	ISR exec.	5.20	104	7.50	150
T3	68	81.60 k	133.80	160.56 k	246	295.20 k	ISR to dec. task	0.65	13	8.15	163
Total	220	264 k	807.80	969.36 k	1449	1828 k	Dec task exec.	5.00	100	13.15	263

**Figure 3.** Delay time distribution for (a) transmission window, (b) decode window, and (c) reception window.

We ascribe the variance of **T2** to the time needed for the OS to detect the interrupt request caused by the doorbell signal, while the variance of **T3** is ascribed to the execution

of the `complete()` function, which is responsible for managing the coordination of threads waiting for an SCMI transaction to end.

4.1.2. LLC End-to-End Latency Analysis

We measured the latency of the SCMI FW module on the LLC, with the same setup, on 1000 `perf_level` set SCMI requests. We leverage performance counters of the CV32E40P [18] core to record the time.

We measure three intervals, namely from the doorbell-triggered ISR to the end of the FreeRTOS-driven SCMI decoding task. The average dispersion of performance counters (5 cycles) has been subtracted from the measurements. We report the results in Table 1b with respect to the 20 frequency of ControlPULP in the HIL's framework. In the table, the first row reports ControlPULP's CLIC latency, [5]. The total LLC decoding time takes 263 clock cycles, i.e., 13.15 μ s @20.

We observe that in the end-to-end SCMI flow, the response time of the LLC accounts for just 1.6% of the total average communication time, negligible when compared with the latencies introduced by the SCMI SW stack on the OSPM. It must be noted that in real silicon, the LLC would run at 500, further reducing this contribution significance. Likewise, we observe that the variability of the entire SCMI communication latency is caused by the Linux stack only, as the LLC variance is, by far, negligible.

As a remark, from the measurements, we note that the limiting factor of SCMI request throughput is (i) **T1**, which determines the speed at which the messages in the mailbox can be processed by the decoding firmware, and (ii) the size of the SW buffer in the SCMI drivers, which collects pending requests and rejects them when full.

4.2. Characterization of End-to-End PM Policy

To assess the impact of SCMI's latency on the control performance, we executed a set of tests on the HIL FPGA framework. We select a square-shaped workload trace with a period of T_{WL} and a 50% duty cycle, alternating between CPU-bound instructions (assumed to run at $f_{T,max}^i$) and memory instructions, whose throughput depends solely on memory subsystem speed.

We compare the proposed SCMI interface with an ideal near-zero latency shared memory transport that bypasses the OS. Additionally, we evaluate multiple HLC configurations and control algorithms for the LLC to assess their impact on performance. The simulated application consists of 500 workload phases of 20 ms each and lasts 10 s when executing at the maximum frequency ($f_{T,max}^i = 3.4$). During the tests, power and thermal capping targets are set and enforced by the LLC FW. When all the co-simulated cores execute at the maximum frequency, both caps are exceeded. The HLC sets the core's frequency to 0.8 during memory phases and the maximum one during compute phases. The tests conducted with a periodic HLC are configured with $T_{ft} = 1$ ms. With the average SCMI communication latency being 70.5 μ s, which is almost an order of magnitude lower than the execution periodicity of both the control task and the communication task ($T_c = 500$ μ s) [5], most of its effects on the control performance are masked in the LLC. For this reason, we restrict the evaluation to $T_{HLC} < T_{WL}$. It must be noted that the simulation time engine is relaxed by a factor of $25\times$ compared to the LLC time resolution. We account for this difference in the emulation setup by artificially delaying the forwarding of new target frequencies to the SCMI channel of 1.7 ms.

In the following, we named control delay (CD) the time elapsed from a workload phase front to an actual change in the applied frequency. We performed three tests: (i) with periodic HLC, (ii) with an event-driven HLC, and (iii) with the proposed LLC optimized FW presented in Section 3.4. The event-driven HLC algorithm used in test (ii) is derived from previous work that optimizes DVFS policies based on workload patterns [6,7]. The average CD for (i) is 2.20 ms for SCMI communication and 1.94 ms for the shared memory case. Test (ii) results in an average CD of 1.35 ms with SCMI communication and 1.16 ms for the shared memory case, showing an improvement thanks to the responsiveness of

the event-driven HLC. Finally, in the proposed optimized LLC FW, in (iii) the average CD reduces to 0.77 ms. This average CD encompasses two distinct scenarios: one in which a workload variation prompts an increase in core frequency, and another where it prompts a decrease. In (iii), distinguishing between these cases, we observe that the CD associated with frequency increases is 1.42 ms, whereas for frequency decreases, the CD is significantly lower, reaching just 114 μ s. Figure 4 shows the distribution of the CD during (i) and (ii), (i) Figure 4a,b, and (ii) Figure 4c,d.

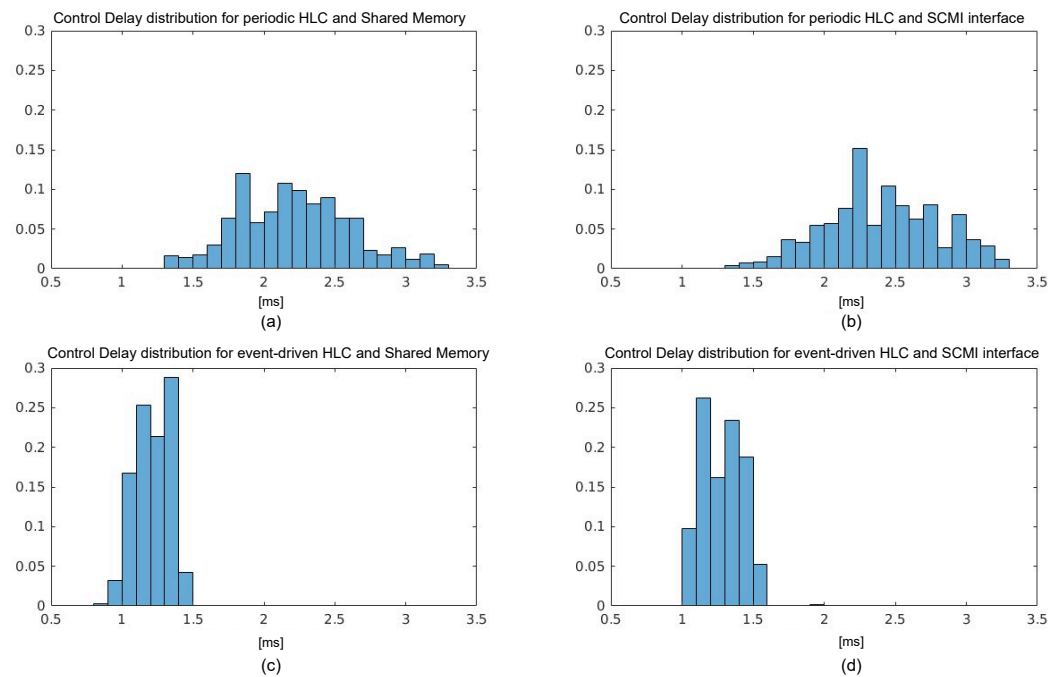


Figure 4. Control Delay distribution for different HLC configurations and interfaces: (a) periodic HLC with shared memory, (b) periodic HLC with SCMI mailbox, (c) event-driven HLC with shared memory, (d) event-driven HLC with SCMI mailbox.

Table 2 reports the control performance for the same three tests measured as the application speedup with regard to periodic HLC and baseline ideal shared memory PMI. From it, we can notice that the event-driven HLC (ii) leads to a speed-up in the application (+2.75%) which further increases with the proposed LLC-optimized FW (+3.18%). Moreover, the CD introduced by the SCMI protocol leads to a marginal reduction in the attained speedup (from -0.02% to -0.3%). These application speedups are due to better energy efficiency, which translates to higher frequencies selected by the LLC FW while enforcing thermal and power caps.

Table 2. Execution time speedup over different HLC and control algorithm configurations.

Configuration	Execution Time Speedup [%]	
	Shared Memory	SCMI
(i) HLC periodic	0.00	-0.02
(ii) HLC event-driven	2.75	2.71
(iii) HLC event-driven, Opt. Control	3.18	2.89

5. Discussion

In the previous section, we first analyzed the end-to-end latency associated with exchanging frequency setpoint messages via the SCMI HW/SW interface. Our analysis reveals that, while only 70.5 μ s at 1.2 are needed for the communication of a new frequency setpoint from the HLC to the HW SCMI mailbox, the highest latency occurs when the OS

waits for an acknowledgment from the LLC's SCMI FW module during the decode window. However, only 13.5 μ s (which is the 2.2% of the total time of the decode window duration) depends on LLC's SCMI FW, suggesting that the majority of the delay happens in the interrupt request handling by the OS. Additionally, the duration of the decode window varies between 83 μ s and 1.2 ms at 1.2 due to variability in interrupt service time.

We then examined how the observed latency affects the control delay and how it can affect the execution time of a reference workload under power and thermal caps, considering the impact of different application-level power management policies (HLC), the power management unit policies (LLC), and the SCMI interface itself. Overall, if the SCMI mailbox operates without congestion, the latency from message transmission to its decoding does not significantly affect the quality of power management control. Instead, the primary critical factors are the policies used in both the application-level power management runtime and the power controller's power management FW. The transition from a time-driven to an event-driven power management led to a decreased latency in the power management command (CD) from around 2.2 ms to 1.35 ms, leading to an application time reduction and energy saving of around 2.7% of the evaluated workload. Similarly, optimizing the LLC control algorithm further reduced CD, with a decrease down to 114 μ s, obtaining a speedup of 2.89%.

These results show that the end-to-end latency of the power management SW stack, from application down to OS drivers and Power Control Unit (PCU) FW, plays a central role in the user's application performance and energy consumption. Based on the characterization mentioned above, we identified that the latency to execute a power management command is dominated by the PCU control policy and not by the communication interface in the ARM SCMI standard. This finding is particularly relevant given the growing interest in the uptake of the NVIDIA ARM Grace architectures in today's data centers.

In large-scale distributed systems, where applications are parallelized across multiple computing nodes, with an alternation in computational and communication phases, lower latency in the power management command propagation enables higher energy saving. As an example, work from Cesarini et al. [19], when evaluating power management runtimes in real data centers based on x86 processors, found that the performance limiting factor of reactive policies was the PCU latency of 500 μ s in Intel systems. Our proposed approach lowers the command propagation latency from 1.35 ms to 114 μ s when reducing the operating frequency. Future work will target optimization in the opposite case when increasing the core's frequency.

Looking ahead, this study has shed light on the key elements of the PM control scheme and, in particular, how the PMI influences end-to-end control quality. We have identified which factors introduce the most significant latency contribution and how this latency can affect workload execution times. Notably, we found that policies at both the HLC and LLC levels have a more substantial impact than the Linux software stack and hardware of the PMI. As newer shared-memory-based PMI, such as RPMI, become more widespread, it is expected that the primary bottleneck in communication latency (including the acknowledgment message) will be the delay introduced by the Linux operating system in handling interrupt routines. In addition, developing core operating point control algorithms that can proactively manage frequencies, without the frequent processing of setpoints from the HLC, could significantly reduce control delay and enhance system efficiency.

6. Conclusions

In this work, we extended the FPGA-based HIL platform introduced in [5], integrating the communication backbone between the HLC and LLC, employing the Linux SCMI SW stack, and implementing an SCMI-MU. These integrations enabled us to characterize the inherent delay of traversing the HW/SW stack, measuring a latency of 70.5 μ s at 1.2, attributed to the dispatching of an SCMI message through the Linux OSPM stack and an average of 603 μ s for processing the SCMI response from the LLC. We then evaluated the impact on end-to-end PM, observing a minimal contribution of the SCMI interface to

the cores' execution time in a nominal periodic HLC simulation. Ultimately, through the optimization of the LLC's control FW, we reduced the latency introduced by the PMI from around 1.3 ms to 114 μ s, resulting in a reduction of the energy consumption and application execution time of approximately 3%.

Author Contributions: Conceptualization, A.d.V., A.B.; methodology, A.d.V. and A.O.; software, A.d.V.; validation, A.d.V., G.B. and A.O.; formal analysis, A.d.V.; investigation, A.d.V.; resources, A.B.; data curation, A.d.V.; writing—original draft preparation, A.d.V. and A.O.; writing review and editing, A.d.V., A.O. and A.B.; visualization, A.d.V.; supervision, A.A. and A.B.; project administration, A.A. and A.B.; funding acquisition, A.B. All authors have read and agreed to the published version of the manuscript.

Funding: This research was partly supported by the EU Pilot for exascale EuroHPC EUPEX (g.a. 101033975), the European Processor Initiative (EPI) SGA2 (g.a. 101036168), the HE EU DECICE project (g.a. 101092582), the HE EU Graph-Massivizer project (g.a. 101093202), and the SPOKE 1: Future HPC & Big Data by PNRR.

Data Availability Statement: The data supporting the findings of this study are included in the article. The code developed for the analysis and design is available in the GitHub repository: <https://github.com/pulp-platform/control-pulp>, accessed on 20 September 2024.

Conflicts of Interest: The authors declare no conflicts of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

Abbreviations

The following abbreviations are used in this manuscript:

ACPI	Advanced Configuration and Power Interface
AP	application-class processors
AVSBUS	Adaptive Voltage Scaling Bus
BMC	Baseboard Management Controller
CD	control delay
CLIC	Core-Local Interrupt Controller
CPPC	collaborative processor performance control
CPU	central processing unit
DSA	domain-specific accelerator
DVFS	dynamic voltage and frequency scaling
FFH	fixed functional hardware
FPGA	field programmable gate array
FW	firmware
GPU	graphic processing unit
HBM	high-bandwidth memory
HIL	hardware in the loop
HLC	high-level controller
HPC	high-performance computing
HW	hardware
ISR	interrupt service routine
LLC	low-level controller
LPI	low-power idle
MSR	model-specific register
OCC	On-Chip Controller
OPAL	OpenPower abstraction layer
OS	Operating System
OSPM	operating system-directed configuration and power management
PCC	platform communication channel
PCF	power control firmware
PCU	power control unit
PE	processing element

PL	Programmable Logic
PLL	phase-locked loop
PM	power management
PMBUS	Power Management Bus
PMCA	programmable many-core accelerator
PMI	power management interface
PS	processing system
PVT	Process, Voltage, Temperature
RAPL	Running Average Power Limit
RTL	Register Transfer Level
RTOS	real-time OS
SCMI	System Control and Management Interface
SCMI-MU	SCMI mailbox unit
SoC	system on chip
SPR	special purpose register
SW	software

References

1. Avgerinou, M.; Bertoldi, P.; Castellazzi, L. Trends in data Centre energy consumption under the European code of conduct for data Centre energy efficiency. *Energies* **2017**, *10*, 1470. [\[CrossRef\]](#)
2. Intel. Power Management in Intel® Architecture Servers. 2009. Available online: https://www.intel.com/content/dam/support/us/en/documents/motherboards/server/sb/power_management_of_intel_architecture_servers.pdf (accessed on 20 September 2024).
3. Grover, A. Modern System Power Management: Increasing Demands for More Power and Increased Efficiency Are Pressuring Software and Hardware Developers to Ask Questions and Look for Answers. *Queue* **2003**, *1*, 66–72. [\[CrossRef\]](#)
4. Arm. Power and Performance Management Using Arm SCMI Specification. 2019. Available online: <https://developer.arm.com/documentation/102886/001?lang=en> (accessed on 20 September 2024).
5. Ottaviano, A.; Balas, R.; Bambini, G.; Del Vecchio, A.; Ciani, M.; Rossi, D.; Benini, L.; Bartolini, A. ControlPULP: A RISC-V On-Chip Parallel Power Controller for Many-Core HPC Processors with FPGA-Based Hardware-In-The-Loop Power and Thermal Emulation. *Int. J. Parallel Program.* **2024**, *52*, 93–123. [\[CrossRef\]](#)
6. Silva, V.R.G.d.; Valderrama, C.; Manneback, P.; Xavier-de Souza, S. Analytical Energy Model Parametrized by Workload, Clock Frequency and Number of Active Cores for Share-Memory High-Performance Computing Applications. *Energies* **2022**, *15*, 1213. [\[CrossRef\]](#)
7. Coutinho Demetrios, A.; De Sensi, D.; Lorenzon, A.F.; Georgiou, K.; Nunez-Yanez, J.; Eder, K.; Xavier-de Souza, S. Performance and energy trade-offs for parallel applications on heterogeneous multi-processing systems. *Energies* **2020**, *13*, 2409. [\[CrossRef\]](#)
8. Kocot, B.; Czarnul, P.; Proficz, J. Energy-aware scheduling for high-performance computing systems: A survey. *Energies* **2023**, *16*, 890. [\[CrossRef\]](#)
9. UEFI. ACPI Specification 6.5. 2023. Available online: <https://uefi.org/specs/ACPI/6.5/> (accessed on 20 September 2024).
10. Arm. Power Control System Architecture. 2023. Available online: <https://developer.arm.com/documentation/den0050/d/?lang=en> (accessed on 20 September 2024).
11. Bartolini, A.; Rossi, D.; Mastrandrea, A.; Conficoni, C.; Benatti, S.; Tilli, A.; Benini, L. A PULP-based Parallel Power Controller for Future Exascale Systems. In Proceedings of the 2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS), Genoa, Italy, 27–29 November 2019; pp. 771–774. [\[CrossRef\]](#)
12. Balas, R.; Ottaviano, A.; Benini, L. CV32RT: Enabling Fast Interrupt and Context Switching for RISC-V Microcontrollers. *arXiv* **2023**, arXiv:2311.08320. [\[CrossRef\]](#)
13. Rosedahl, T.; Broyles, M.; Lefurgy, C.; Christensen, B.; Feng, W. Power/Performance Controlling Techniques in OpenPOWER. In *High Performance Computing, Proceedings of the ISC High Performance 2017, Frankfurt, Germany, 18–22 June 2017*; Kunkel, J.M., Yokota, R., Taufer, M., Shalf, J., Eds.; Springer: Cham, Switzerland, 2017; pp. 275–289.
14. Arm. SCP-Firmware—Version 2.13. 2023. Available online: <https://github.com/Arm-software/SCP-firmware> (accessed on 20 September 2024).
15. Arm. Arm Cortex-A75 Technical Reference Manual. 2024. Available online: <https://developer.arm.com/documentation/ka005129/latest/> (accessed on 20 September 2024).
16. Patterson, D.A.; Hennessy, J.L. *Computer Organization and Design*, 2nd ed.; Morgan Kaufmann Publishers: Burlington, MA, USA, 1998; p. 715.
17. Bartolini, A.; Cacciari, M.; Tilli, A.; Benini, L. Thermal and Energy Management of High-Performance Multicores: Distributed and Self-Calibrating Model-Predictive Controller. *IEEE Trans. Parallel Distrib. Syst.* **2013**, *24*, 170–183. [\[CrossRef\]](#)

18. OpenHW Group. CV32E40P: In-Order 4-Stage RISC-V CPU Based on RI5CY from PULP-Platform. 2024. Available online: <https://github.com/openhwgroup/cv32e40p> (accessed on 20 September 2024).
19. Cesarini, D.; Bartolini, A.; Borghesi, A.; Cavazzoni, C.; Luisier, M.; Benini, L. Countdown slack: A run-time library to reduce energy footprint in large-scale MPI applications. *IEEE Trans. Parallel Distrib. Syst.* **2020**, *31*, 2696–2709. [[CrossRef](#)]

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.