ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

DOTTORATO DI RICERCA IN

INGEGNERIA ELETTRONICA, TELECOMUNICAZIONI E TECNOLOGIE DELL'INFORMAZIONE

Ciclo 38

Settore Concorsuale: 09/H1 - SISTEMI DI ELABORAZIONE DELLE INFORMAZIONI

Settore Scientifico Disciplinare: ING-INF/05 - SISTEMI DI ELABORAZIONE DELLE INFORMAZIONI

POWER-THERMAL MODELING AND MANAGEMENT OF MODERN CHIPLET-BASED PROCESSORS

Presentata da: Antonio Del Vecchio

Coordinatore Dottorato

Davide Dardari

Supervisore

Andrea Bartolini

Co-supervisore

Andrea Acquaviva

Esame finale anno 2026

Abstract

As the semiconductor industry moves beyond the limits of monolithic integration, **chiplet-based architectures** have become a cornerstone of continued performance and energy-efficiency scaling in high-performance computing (HPC) processors. By partitioning large dies into smaller, specialized chiplets interconnected through high-bandwidth die-to-die links, designers can overcome reticle-size and yield limitations while enabling heterogeneous integration of compute, memory, and I/O subsystems within a single package. However, this architectural shift introduces new challenges for **power and thermal management**. Thermal coupling across chiplets causes non-uniform temperature distributions and localized hotspots; leakage currents and package-level heat transfer exacerbate power density; and performance degradation can occur when protective throttling mechanisms are triggered. Efficient operation therefore requires coordinated and low-latency control among multiple on-die Power-Management Subsystems (PCS) distributed across chiplets.

Recent research has explored hierarchical power-control architectures, standardized management protocols such as ACPI and SCMI, and data-driven thermal modeling to improve runtime efficiency in multi-chiplet systems. However, these approaches remain limited by proprietary PCS implementations, kernel-mediated communication paths, and the lack of empirical modeling techniques that capture the thermal behavior of chiplet-based processors.

This dissertation addresses these challenges through four complementary research contributions spanning hardware architecture, communication interfaces, and data-driven modeling.

The first contribution introduces **ControlPULP**, an open and scalable RISC-V–based PCS designed to execute fine-grained control loops at sub-millisecond time scales. Its parallel microcontroller architecture and DMA-assisted sensing enable reproducible research on distributed, high-performance power and thermal control, marking a step toward open and programmable control planes for next-generation processors. This contribution directly addresses the lack of open, research-grade PCS architectures capable of scaling with core counts and supporting sub-millisecond control loops.

The second contribution provides a **fine-grained characterization of standardized power-management interfaces** in modern HPC processors. Using an FPGA-based hardware-in-the-loop framework and a fully compliant Linux SCMI stack, the study reveals latency bottlenecks across software, firmware,

and hardware layers. Measured end-to-end delays reach up to 1.3 ms per control cycle, while optimized firmware reduces the control delay to 114 μs, yielding up to 3% shorter execution time. This analysis addresses the previously unquantified latency behavior of delegation-based control protocols, clarifying their real-time limitations and optimization potential.

The third contribution proposes a **user-space communication substrate for distributed PCS networks**, based on the Micro XRCE-DDS middleware. A new shared-memory transport layer eliminates kernel-mediated bottlenecks and enables low-latency, peer-to-peer coordination among PCS nodes and a host processor. The publish/subscribe abstraction allows dynamic addition of control topics and application-level policies, achieving an average Client-side processing time of about 490 μs at 20 MHz on the FPGA prototype—corresponding to only a few tens of microseconds in a realistic silicon implementation—with predictable memory usage below 2 KiB per topic. This contribution overcomes the rigidity and kernel dependence of current control planes, providing a scalable substrate for distributed power-management communication.

The fourth contribution develops a **compact empirical thermal–power modeling methodology** for many-core and chiplet-based processors. By combining performance-counter-based power estimation with distributed MISO-ARX thermal identification, the approach captures both intra- and cross-chiplet thermal interactions directly from silicon measurements and reconstructs approximate processor floorplans to reduce model complexity. Validation on a 56-core Intel Sapphire Rapids processor demonstrates power-prediction RMSE of 2.464 W and thermal-prediction MSE of 3.756 $^\circ C^2$, suitable for predictive control. This contribution extends data-driven thermal–power modeling techniques to multi-chiplet architectures, enabling compact, measurement-based models for predictive control in heterogeneous packages.

Together, these results form a coherent framework that advances the state of the art in open, distributed, and predictive power and thermal management for next-generation many-core and chiplet-based processors. The proposed architectures and methodologies lay the groundwork for future research toward standardized, adaptive, and energy-aware control frameworks across heterogeneous HPC platforms.

Acknowledgments

I would first like to express my sincere gratitude to my supervisor, Andrea Bartolini, for guiding me throughout this doctoral journey. I am deeply thankful for his constant support, availability, and for giving me the opportunity to explore new research directions and engage in our frequent and enriching theoretical discussions.

I am also grateful to my co-supervisor, Andrea Acquaviva, for his guidance and continuous support, and for contributing, together with Andrea Bartolini, to making it possible for me to pursue my PhD within the ECS Lab, an environment in which I have grown both professionally and personally.

My sincere thanks go to Professor Roberto Diversi for his availability and for generously sharing his expertise in thermal modeling during the final year of my doctoral studies.

I would like to thank Carlo Cavazzoni, Antonio Sciarappa, and Carolina Berucci for giving me the opportunity to spend a research period at Leonardo Labs.

I am particularly grateful to Professor Luca Benini for giving me the opportunity to carry out a research period at ETH Zurich and for his guidance throughout that experience.

I also wish to thank all my project collaborators and all the members of the ECS Lab for their support, collaboration, and for contributing to a stimulating and inspiring research environment.

Finally, I would like to thank everyone who has supported me over these years, who has offered advice and encouragement, and who has helped make this intense journey lighter and more meaningful.

List of Publications

The following peer-reviewed publications are directly related to the research work presented in this dissertation:

1. A. Ottaviano, R. Balas, G. Bambini, A. Del Vecchio, M. Ciani, D. Rossi, L. Benini, and A. Bartolini, **"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,"** *International Journal of Parallel Programming*, vol. 52, no. 1, pp. 93–123, Springer, 2024.

   Author's contribution: The candidate contributed to the hardware/software co-design of the FPGA-based HIL platform by designing and integrating the AXI4 interconnect architecture between the Processing System and ControlPULP, enabling bidirectional communication and mailbox-based communication. The candidate also developed the FreeRTOS-based firmware library for SCMI message decoding, which enabled the latency evaluations reported in the paper.

2. A. Del Vecchio, A. Ottaviano, G. Bambini, A. Acquaviva, and A. Bartolini, **"Performance Characterization of Hardware/Software Communication Interfaces in End-to-End Power Management Solutions of High-Performance Computing Processors,"** *Energies*, vol. 17, no. 22, article 5778, MDPI, 2024.

   Author's contribution: The candidate led the implementation and integration of the SCMI communication infrastructure within the HIL framework, including hardware design and implementation of the Linux-compliant SCMI Mailbox Unit, Linux driver configuration, and system-level integration on the ZCU102 platform. The candidate also performed the experimental latency characterization of the SCMI software stack and firmware decoding mechanisms.

Additional contributions are presented throughout the dissertation.

Contents

Abstract iii

Acknowledgements v

List of Publications vii

List of Figures xv

List of Tables xvii

List of Acronyms xxiii

1 Introduction 1
- 1.1 Power and Thermal Management in Chiplet-Based Architectures 3
- 1.2 State of the Art . . . 4
  - 1.2.1 Power-Management Infrastructure in Modern HPC Processors . . . 4
  - 1.2.2 Thermal–Power Modeling for Predictive Control Algorithms 5
- 1.3 Motivation . . . 6
- 1.4 Contributions and Outline . . . 8

2 Background & Fundamentals 11
- 2.0.1 Transistor Scaling and Power Consumption Considerations 11
- 2.0.2 From Transistor Physics to System-Level Power Management 12

3 ControlPULP: A RISC-V On-Chip Parallel Power Controller for HPC 15
- 3.1 Introduction . . . 15
- 3.2 Related Work . . . 18
- 3.3 The ControlPULP platform . . . 21
  - 3.3.1 Controlled plant and Power Control Firmware . . . 21
  - 3.3.2 System architecture . . . 23
- 3.4 FPGA-based HIL thermal, power, performance and monitoring emulation framework . . . 27
  - 3.4.1 HIL system and PCS mapping on FPGA . . . 27

3.4.2 Plant simulation . . . 28
3.4.3 HIL testing procedure . . . 30
3.5 Evaluation . . . 31
3.5.1 Area evaluation . . . 31
3.5.2 Firmware *control action* . . . 33
3.5.3 *In-band* Services . . . 33
3.5.4 System-level PCF step evaluation . . . 36
3.5.5 Control-level PCF evaluation . . . 37
3.5.6 Standalone QoS evaluation on the HIL FPGA-SoC framework 38
3.6 Conclusion . . . 43

4 Performance Characterization of Power-Management Interfaces in HPC Processors 45
4.1 Introduction . . . 45
4.2 Background and Related Works . . . 47
4.2.1 Overview and Terminology . . . 47
4.2.2 HLC Components . . . 49
4.2.3 LLC Components . . . 49
4.2.4 HLC-LLC Interface . . . 50
4.3 Methodology: HIL Framework on FPGA . . . 51
4.3.1 SCMI-MU Implementation . . . 52
4.3.2 Linux SCMI Software Stack . . . 52
4.3.3 LLC SCMI Firmware Module . . . 53
4.3.4 LLC Power Management Policy Optimized for Latency . 54
4.3.5 HLC Policy . . . 54
4.4 Evaluation . . . 55
4.4.1 SCMI Latency Characterization . . . 55
4.4.2 Characterization of End-to-End Power Management Policy 58
4.5 Discussion . . . 60
4.6 Conclusions . . . 62

5 Towards User-Space Power-Management Communication Interfaces 63
5.1 Introduction . . . 63
5.2 Related Works . . . 65
5.2.1 Kernel-mediated power-thermal interfaces . . . 65
5.2.2 Inter-controller synchronization in multi-chiplet systems . 65
5.2.3 Micro XRCE-DDS . . . 66
5.2.4 Summary . . . 66
5.3 Methodology . . . 66
5.3.1 Platform Overview . . . 66
5.3.2 Shared-Memory Transport . . . 67
5.4 Evaluation . . . 68
5.4.1 Software Setup . . . 68
5.4.2 Firmware Execution Time . . . 69
5.4.3 Stack Usage Measurement . . . 70

5.4.4 Shared–Memory Occupation and Protocol Overhead . . . 71
5.5 Conclusions . . . 72

6 Compact Thermal and Power Models for Many-core chiplet-based Architectures 75
6.1 Introduction . . . 75
6.2 Related Work . . . 77
6.2.1 Thermal Modeling . . . 77
6.2.2 Floorplan Inference From Measurements . . . 78
6.2.3 Power Modeling From Performance Counters . . . 78
6.3 Methodology . . . 78
6.3.1 Workloads . . . 78
6.3.2 Monitoring and processing . . . 79
6.3.3 Thermal Model . . . 81
6.3.4 Power Model . . . 81
6.3.5 Processor Floorplan Inference . . . 82
6.4 Evaluation . . . 85
6.4.1 Experimental Setup . . . 85
6.4.2 Power Model . . . 85
6.4.3 Thermal Model . . . 87
6.4.4 Processor Floorplan Inference . . . 89
6.5 Conclusion . . . 91

7 Context and Technological Framework 95

8 Conclusions and Future Work 97

List of Figures

3.1 High-level overview of the system. We highlight on-chip and off-chip high-level controllers (HLCs) (Operating System (OS) and Baseboard Management Controller (BMC)), the low-level controller (LLC) (ControlPULP) and the multiple-input multiple-output (MIMO) IO interfaces. Furthermore, the figure details the power control firmware (PCF) phases described in Sec. 3.3.1 . . 21

3.2 ControlPULP software stack. The application control policy (PCF) executes on top of FreeRTOS, which controls the hardware with target-specific drivers and hardware abstraction layer (HAL) application programming interfaces (APIs) 22

3.3 ControlPULP hardware architecture. On the left, the *manager domain* with the *manager core* and surrounding peripherals. On the right, the *cluster domain* accelerator with the eight cores (workers) 24

3.4 hardware in the loop (HIL) test procedure on an field-programmable gate array (FPGA)-system on chip (SoC) with ControlPULP . . 28

3.5 ControlPULP RTL testbench simulation environment 32

3.6 **(a)-(b)** Firmware *control action*, execution time, and speedup comparison between single-core (*manager domain*) and 8-cores (*cluster domain*). **(c)** *in-band* data acquisition from simulated Process, Voltage, Temperature (PVT) registers, execution time without and with direct memory access (DMA) in 1-D and 2-D configurations. **(d)** Execution time in the interrupt handler from the interrupt edge to its completion with a basic System Control and Management Interface (SCMI) message at varying interconnect network access latency to the mailbox 35

3.7 Benefits of a performant parallel power controller system (PCS) on the control problem, namely, room for more computationally-intensive control policies from the concurrent acceleration. The figure shows that the hyper-period free time window (*slack*) increases to almost 95% when the policy is accelerated with the *cluster domain* 37

3.8 Thermal, power capping and dynamic voltage and frequency scaling (DVFS) emulation on the FPGA-SoC HIL framework with `Wsynth`. Requests coming from the HLCs (OS or BMC) governors, such as the target frequency and power budget, are processed by the reactive control, phases (P6)-(P7) described in Sec. 3.3.1. The number of controlled cores is 72. 39
3.9 HIL and model in the loop (MIL) comparison when `Wsynth` is assigned to the controlled system. The emulation assumes a floorplan with a tile of 9 cores and runs for $2s$ to align with MATLAB runtime. 40
3.10 Comparison between the modified IBM OpenPOWER control with per-core temperature proportional integral derivative (PID) for frequency reduction and the original IBM OpenPOWER control. The simulation time is $2s$ 42
3.11 Comparison between the PCF control and the modified IBM OpenPOWER firmware with per-core temperature PID for frequency reduction. The simulation time is $2s$ 42

4.1 Overview of the main power management (PM) components of modern high performance computing (HPC) SoCs. The figure highlights the power management interfaces (PMIs) linking HLCs to the LLCs from a hardware (HW) and software (SW) perspective in Arm-based server-class central processing units (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 chapter. 48
4.2 Sequence of function calls within the Linux CPUFreq stack, for a perf_level_set SCMI request. 53
4.3 Delay time distribution for (**a**) transmission window, (**b**) decode window, and (**c**) reception window. 56
4.4 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. 60

5.1 Micro eXtremely Resource Constrained Environments - Data Distribution Service (Micro XRCE-DDS) implementation overview on Xilinx ZCU102 board. 67
5.2 Shared Memory Layout for XRCE custom transport implementation. 68

6.1 Monitoring and processing framework architecture overview. . . . 79
6.2 Performance monitoring software architecture overview. 80
6.3 Average processor power consumption over different configurations of active cores for each chiplet. 86
6.4 Idle power vs. temperature dependency. 88

6.5 Global thermal coupling coefficient map in logarithmic scale. Each row index identifies the core for which the thermal model is derived, whereas each column index denotes the core whose exogenous thermal coefficients contribute to the modeled temperature, producing the value reported in the heatmap. 89
6.6 Inferred processor floorplan showing the four chiplets (C0–C3) and the mapping of core identifiers to physical positions. Core IDs label active cores, "MC" indicates memory controllers, and "R" denotes redundant cores. 90
6.7 Distribution of Mean Squared Error (MSE) on prediction test for different input configurations of the thermal model. 92

List of Tables

3.1 Comparison among existing proprietary and freely-available PCS from industry and academia. 19
3.2 ControlPULP resource utilization on the Xilinx UltraScale+ ZCU102. 28
3.3 ControlPULP post-synthesis area breakdown on GF22FDX technology. 32
3.4 Interrupt latency from interrupt edge to the first instruction in the interrupt handler as number of cycles. 34
3.5 Execution time T of a periodic frequency control task (PFCT) step, single-core and cluster configurations. SCMI commands exchange and off-die transfers, handled by the SoC *manager core*, are not included in the comparison since they are a shared overhead. 38

4.1 SCMI call stack time measured at 1.2 GHz. 57
4.2 ControlPULP SCMI decoding time @20 MHz. 57
4.3 Execution time speedup over different HLC and control algorithm configurations. 60

5.1 Timing characterization of Client execution phases (ControlPULP @ 20 MHz). 70
5.2 Peak stack usage and incremental cost per added topic. 71
5.3 Protocol overhead as a function of payload size. 72

6.1 Selected input metrics and regression coefficients for the power model. 87

List of Acronyms

ACPI Advanced Configuration and Power Interface. 3–6, 13, 46, 48, 50, 64, 65

AI Artificial Intelligence. 1, 75

AP application-class processor. 16, 17, 45, 46, 49

API application programming interface. xiii, 22

APU Application Processing Unit. 30

AVSBUS Adaptive Voltage Scaling. 22, 27, 49

BMC Baseboard Management Controller. xiii, xiv, 5, 16, 18, 20–23, 27, 29, 38, 39, 48, 64

CD control delay. 59, 61

CLIC core-local interrupt controller. 24–26, 31, 34, 52, 58

CLINT core-local interruptor. 24, 34

COTS commercial off-the-shelf. 20

CPPC collaborative processor performance control. 50

CPU central processing unit. xiv, 2, 4–6, 16–23, 26–29, 32, 33, 36, 37, 39, 43, 46, 48, 49, 51, 55, 78, 85, 86, 95

CSR Control and Status Register. 24, 69

D2D Die-to-Die. 2, 66

DDR double data rate. 66

DDS Data Distribution Service. 66

DFS dynamic frequency scaling. 22, 29

DMA direct memory access. xiii, 18, 19, 24, 25, 31, 34, 35, 43, 97

DRAM dynamic random access memory. 66, 67, 73

DSA domain-specific accelerator. 49

DTM dynamic thermal management. 15, 16

DVFS dynamic voltage and frequency scaling. xiv, 3, 6, 8, 13, 15, 18, 23, 34, 38, 39, 46, 49, 50, 59

DVS dynamic voltage scaling. 22, 27, 29

FAME FPGA Architecture Model Execution. 20

FFH fixed functional hardware. 50

FIL FPGA in the loop. 27

FPGA field-programmable gate array. xiii, xiv, 17, 20, 27, 28, 30, 31, 37, 39, 43, 46, 51, 55, 58, 66, 72, 97

FPU floating point unit. 19, 25

FSBL First Stage BootLoader. 30

FW firmware. 3–5, 8, 46, 49, 50, 53–55, 58, 59, 61, 62

GPU graphics processing unit. 1, 16, 49, 98

HAL hardware abstraction layer. xiii, 22

HBM high-bandwidth memory. 49

HeSoC heterogeneous system on chip. 17, 18, 27, 43

HIL hardware in the loop. xiii, xiv, 8, 18, 20, 27–29, 38–40, 46, 47, 51, 55, 58, 62, 97

HLC high-level controller. xiii, xiv, 16, 20–23, 26, 35, 38–40, 46–55, 58–62

HPC high performance computing. xiv, 4, 6, 15, 21, 22, 26, 29–32, 34, 37, 41, 45–49, 63, 75, 78, 86, 95, 97, 98

HW hardware. xiv, 16–20, 30, 31, 37, 43, 46–53, 60, 62–66

I/O Input/Output. 2

IP intellectual property. 2, 19, 27

IPC instructions per cycle. 29

ISA Instruction Set Architecture. 64, 95

ISR interrupt service routine. 52, 54, 58

ISUT integrated system under test. 17, 27, 28, 31

LLC low-level controller. xiii, xiv, 4, 8, 16, 17, 20–22, 27, 46–55, 58, 59, 61–63

LLM Large Language Model. 75

LPI Low Power Idle. 50

MC Memory Controller. 82–84

MCP Manageability Control Processor. 4, 19

MCTP Management Component Transport Protocol. 22, 27, 29

MCU microcontroller unit. 4, 16, 17, 20, 66

MHU message handling unit. 47

Micro XRCE-DDS Micro eXtremely Resource Constrained Environments - Data Distribution Service. xiv, 64–67, 72, 97, 98

MIL model in the loop. xiv, 18, 20, 31, 39, 40

MIMO multiple-input multiple-output. xiii, 8, 16, 17, 21

MISO-ARX Multiple-Input Single-Output AutoRegressive model with eXogenous inputs. 76, 77, 81, 87, 88, 90, 91, 98

MPC Model Predictive Control. 76

MPSoC multi-processor system on chip. 17, 20, 30, 37, 38

MQTT Message Queuing Telemetry Transport. 30

MSE Mean Squared Error. xv, 77, 90–93

MSR model-specific register. 50, 51, 63, 80

MTU Maximum Transmission Unit. 69, 71, 72

NoC network on chips. 22, 35, 36

NUMA Non-Uniform Memory Access. 85, 88

OCC On-Chip Controller. 4, 19, 26, 41, 50

OCM on-chip memory. 27

OPAL OpenPower abstraction layer. 50

OPP Operating Performance Point. 13

OS Operating System. xiii, xiv, 4, 8, 13, 16, 18–21, 23, 26, 29, 35, 36, 38, 39, 45, 46, 49–51, 57, 58, 61

OSPM operating system-directed configuration and power management. 3, 4, 45–52, 58, 62–64

PCC platform communication channel. 50

PCF power control firmware. xiii, xiv, 16–23, 26, 27, 31, 33, 35–37, 41, 42, 53, 54

PCS power controller system. xiii, xvii, 3–6, 8, 16–21, 26, 27, 29–32, 36, 37, 63, 64, 66–68, 73, 95, 97

PCU Power Control Unit. 4, 18, 61

PE processing element. 15, 16, 18–23, 25, 26, 32–34, 38, 39, 41, 48–50, 76

PFCT periodic frequency control task. xvii, 22, 23, 35, 36, 38

PID proportional integral derivative. xiv, 19, 23, 38, 41, 42, 54

PIL processor in the loop. 27

PL Programmable Logic. 27, 30, 51, 66

PLDM Platform Level Data Model. 27, 29

PLIC platform-level interrupt controller. 34

PLL phase locked loop. 3, 13, 36, 37, 54

PM power management. xiv, 45–51, 54, 55, 61, 62

PMBUS Power Management Bus. 22, 27, 49

PMCA programmable many-core accelerator. 17, 18, 49

PMI power management interface. xiv, 46–49, 51, 54, 55, 59, 61, 62

PMU Performance Monitoring Unit. 80

PRBS Pseudo-Random Binary Sequence. 79, 81

PS Processing System. 27, 30, 38, 51, 52, 55, 66

PVCT periodic voltage control task. 22, 23, 35, 39

PVT Process, Voltage, Temperature. xiii, 3, 13, 15, 18, 19, 22, 23, 25, 26, 31, 33–35, 48, 64

QAP Quadratic Assignment Problem. 82, 84

QoS quality of service. 18, 31, 37, 66

RAM Random Access Memory. 85

RAPL Running Average Power Limit. 50

RMSE Root Mean Square Error. 87, 91

RPMI RISC-V Power Management Interface. 3, 13, 50, 51, 62, 64, 96, 98

RTL register transfer level. 20, 31, 35, 51, 66

RTOS real-time OS. 17, 49, 66

RTPS Real-Time Publish Subscribe. 66, 69

SCMI System Control and Management Interface. xiii, xvii, 3–6, 13, 19, 22, 26, 29, 31, 35, 36, 38, 46, 47, 50–55, 57–62, 64, 65, 95–98

SCMI-MU SCMI mailbox unit. 47, 51–54, 62

SCP System Control Processor. 4, 19

SGD Stochastic Gradient Descent. 81

SHV selective hardware vectoring. 24

SIL software in the loop. 18, 20

SIMD single instruction, multiple data. 29

SMP symmetric multiprocessing. 17, 30

SMU System Management Unit. 4, 18, 19

SoA state-of-the-art. 20, 23, 25, 26, 31

SoC system on chip. xiii, xiv, xvii, 17, 19, 21, 28, 31, 34, 37–39, 46–49, 51

SPR special purpose register. 50, 51

STM static thermal management. 15

SW software. xiv, 16–20, 30, 31, 37, 43, 46–48, 51, 53, 54, 58, 60, 62, 63, 66, 69

TDP thermal design power. 2, 15, 16, 39

TPU Tensor Processing Unit. 1

VRM voltage regulator module. 3, 13, 16, 21–23, 27, 41, 64

XML Extensible Markup Language. 71

Chapter 1

Introduction

Until 2005, Moore's law [79] combined with Dennard scaling [39] sustained an exponential growth in processor performance. Increasing transistor density automatically delivered higher clock frequencies at nearly constant power density, allowing monolithic single-core designs to scale almost effortlessly. Performance gains were thus largely achieved through technology scaling rather than architectural innovation.

Around the mid-2000s, the breakdown of Dennard scaling slowed the reduction of supply voltage and forced the industry to shift toward multicore architectures. Instead of relying on ever-faster clock speeds, performance growth had to come from integrating a steadily increasing number of cores per die. This shift made software-level parallelism essential to fully exploit the available hardware.

However, because the supply voltage could no longer decrease proportionally, integrating more cores on the same die raised the overall power density. Processors soon hit the so-called *power wall* [46] — the practical limit of performance scaling imposed by thermal-design and power-delivery constraints. With voltage scaling stalled, energy efficiency worsened and clock frequencies plateaued, motivating the transition to parallelism as the main path to performance scaling.

The *dark-silicon* phenomenon [46] refers to portions of the die that must remain powered off or clock-gated to keep the processor within its thermal and power budgets. Initially, this limitation was driven not only by power-capping restrictions but also by the lack of sufficiently parallel applications able to use all on-die cores efficiently.

A decade later, the rise of deep-learning workloads [65] created a new class of massively parallel applications that saturate both the power and thermal envelopes of monolithic processors. To sustain these workloads, the industry turned to *heterogeneous computing platforms* that integrate specialized accelerators — such as graphics processing units (GPUs) [89], Tensor Processing Units (TPUs) [61], or custom Artificial Intelligence (AI) engines — optimized for matrix-intensive kernels. In this context, dark silicon also manifests as specialized hardware blocks that remain powered off until invoked by suitable workloads.

Modern high-performance processors now face physical and manufacturing

barriers that limit the scalability of single-die designs. Increasing the core count on a single die exacerbates routing congestion and signal-integrity issues. On advanced 7–5 nm nodes, monolithic dies exceeding roughly 600 mm^2 approach both the photolithography reticle-size limit [91] and yield-related costs that become economically prohibitive. Large dies also develop severe thermal hotspots that are difficult to cool efficiently [110], reducing reliability and sustained performance.

To overcome these obstacles, the industry adopted ***chiplet-based*** architectures [35]. Instead of a single large die, multiple chiplets—each typically integrating a balanced set of cores, cache slices, memory controllers, and Input/Output (I/O) blocks—are placed side by side on a silicon interposer. The interposer provides high-bandwidth, low-latency Die-to-Die (D2D) links between chiplets, improving manufacturing yield and reducing local power density. Prominent examples include AMD's EPYC Genoa CPUs (up to 96 Zen 4 cores distributed across 12 CCD chiplets plus a central I/O die) [4] and Intel's Sapphire Rapids Xeon processors (up to 60 Golden Cove cores integrated in 4 compute-tile chiplets plus a base I/O die) [35].

Beyond partitioning large monolithic dies, chiplet-based architectures represent a fundamental paradigm shift in processor design and mark the transition toward the so-called *SysMoore era* [111]. In this new phase, continued performance scaling is no longer achieved solely through transistor miniaturization—as predicted by Moore's Law—but through *system-level integration* of heterogeneous components within the same package. By assembling multiple chiplets, possibly fabricated in different process nodes or by different vendors, designers can integrate specialized computing, memory, and I/O subsystems while maintaining flexibility and high manufacturing yield.

This modular design approach provides several tangible benefits: (i) *cost reduction*, since smaller chiplets improve yield and can be independently optimized for their respective functions; (ii) *design modularity*, enabling reuse of existing intellectual property (IP) and rapid exploration of architectural variants; and (iii) *faster time-to-market*, as pre-validated chiplets can be recombined into new system configurations without requiring full redesign of the entire die. Overall, chiplet-based integration extends the scalability of modern processors beyond the limits of traditional Moore's Law scaling and opens new opportunities for heterogeneous, domain-specific computing within a single package.

While chiplet-based integration brings substantial advantages in scalability, cost efficiency, and design flexibility, it also introduces new challenges at the physical and architectural levels. The power density of state-of-the-art processors remains extremely high, and the total thermal design power (TDP) budget for high-end CPUs still lies in the 200–400 W range [71]. Distributing cores across multiple smaller chiplets mitigates local manufacturing and thermal-density issues, but it does not eliminate overall power and thermal constraints. Significant heat coupling persists both within and across chiplets, as the silicon interposer and package materials still conduct heat between active regions [86]. Local hotspots can therefore develop even under balanced workloads, and the die-to-die interconnect fabric itself introduces additional latency and power overhead.

Furthermore, the increasingly complex package geometry complicates power delivery and heat dissipation throughout the package. As a result, *power and thermal management* remain central design concerns, critical to achieving reliable and energy-efficient performance in modern chiplet-based high-performance processors.

1.1 Power and Thermal Management in Chiplet-Based Architectures

State-of-the-art power and thermal management in high-performance processors relies on techniques such as dynamic voltage and frequency scaling (DVFS), turbo-boost modes, clock- and power-gating, and emergency thermal throttling [110], [69], [105], [50], [53]. These techniques aim to select operating points — voltage/frequency pairs — for multiple power domains to maximize performance and energy efficiency while staying within safe limits of temperature, power, and current.

A typical processor features a network of on-chip sensors — such as PVT sensors — and distributed actuators, including on-chip phase locked loops (PLLs) for frequency control and off-chip voltage regulator modules (VRMs) for power delivery. At the heart of this infrastructure lies the *PCS*, a dedicated microcontroller running firmware (FW) that executes the control policies. The PCS collects sensor data, computes the appropriate operating points, and drives the actuators accordingly.

The operating system-directed configuration and power management (OSPM) layer — often referred to as the governor in Linux-based systems — interacts with the PCS through standardized interfaces (e.g., Advanced Configuration and Power Interface (ACPI) [117], SCMI [73], or RISC-V Power Management Interface (RPMI) [101]) by requesting performance levels or power caps for specific domains.

In chiplet-based processors, the same basic techniques are used, but the control problem becomes substantially more complex. Each chiplet exhibits its own thermal profile: equal power allocations can lead to uneven temperature distributions. Heat propagates through the silicon interposer [86], causing *leakage-induced power rise* — the exponential increase of static power with temperature [68] — thereby worsening hotspots in neighboring chiplets. Throttling triggered by an overheated core in one chiplet can also degrade the performance of the whole processor. The die-to-die communication fabric itself adds power overhead that must be accounted for in the control loop.

To address these challenges, modern designs often replace a single centralized PCS with a *distributed approach*, deploying one PCS per chiplet. Each PCS governs a subset of the chip's power domains locally; while this simplifies local control, it complicates global coordination because system-level thermal and power policies now require a dedicated communication layer among PCSs to share state information and enforce consistent global decisions.

1.2 State of the Art

This dissertation focuses on two complementary aspects of power and thermal management in modern HPC processors: (i) the **power-management infrastructure**—the hardware/software organization of PCSs and their communication interfaces with the host OS; and (ii) the **thermal–power modeling techniques** required by advanced PCSs to predict temperature dynamics under varying workloads.

1.2.1 Power-Management Infrastructure in Modern HPC Processors

Modern HPC processors integrate tens to hundreds of cores and rely on sophisticated power- and thermal-management strategies to operate within strict power-density and reliability limits. Traditional monolithic CPUs typically embed a single PCS in the uncore domain, implemented as a microcontroller-class unit executing reactive control loops. Commercial examples include Intel's Power Control Unit (PCU) [105] [118], AMD's System Management Unit (SMU) [2], IBM's On-Chip Controller (OCC)[102],[55], and Arm's dual-core System Control Processor (SCP)/Manageability Control Processor (MCP) [6] control subsystem. These controllers, however, are proprietary and expose little about their internal hardware/software partitioning, which limits reproducibility and research on advanced closed-loop policies [88].

Scalability limits of current PCS solutions.

Existing PCSs generally follow the same design pattern: a single embedded microcontroller unit (MCU) running a FW control loop, sometimes assisted by simple hardware state machines or small accelerators. Such designs are adequate for modest core counts but become difficult to scale to the hundreds of cores found in recent HPC processors, or to support the sub-millisecond loops demanded by advanced predictive control.

Latency in power-management interfaces.

Modern HPC platforms often adopt a delegation-based power-management model in which a high-level OSPM cooperates with a low-level controller (LLC)—representing the platform-side power-management logic, typically implemented by the on-chip power-control subsystem (PCS)—through standardized interfaces such as ACPI [117] or SCMI [12]. In this scenario, the quality of closed-loop power control—intended as the controller's ability to regulate core frequency and voltage within thermal and power constraints while minimizing performance degradation—depends not only on the control policy itself but also on the latency of the communication backbone that links the OSPM with the LLC FW. To the best of the author's knowledge, no prior work has provided a quantitative characterization of the latency introduced by power-management

interfaces. This latency—arising from message dispatch, protocol overheads, and FW response time—can, however, significantly affect the responsiveness of power-management loops, especially under rapidly varying workloads.

Control-interface fragmentation and distributed PCSs.

In monolithic systems, a single PCS communicates with the operating system or with a board-level BMC through standardized interfaces such as ACPI or SCMI. As processors evolve toward *chiplet-based* architectures [83, 3, 4, 35], each chiplet typically integrates a local controller managing its own power domains, leading to a distributed control hierarchy. Coordinating multiple PCSs across chiplets requires scalable and low-latency communication channels for telemetry and control exchange.

Current standards like *Arm SCMI* [12] were designed for centralized agent–platform models and lack native primitives for peer-to-peer synchronization or cross-die coordination. Moreover, existing implementations are tightly coupled to kernel drivers and version-specific stacks, hindering extensibility and rapid prototyping. This motivates the exploration of flexible communication substrates that decouple the power-management control plane from kernel mediation and enable scalable coordination among distributed controllers.

1.2.2 Thermal–Power Modeling for Predictive Control Algorithms

The transition from monolithic multicore CPUs to heterogeneous chiplet-based packages has fundamentally changed the requirements of thermal–power modeling [120].

Two main modeling families dominate the literature.

(i) *Simulation-driven gray-box models* combine detailed floorplans with RC-network solvers and material properties to reproduce temperature dynamics with sub-degree accuracy [82]. However, these methods depend on proprietary design-time information that is seldom available for commercial processors and whose fidelity degrades with package-to-package variability, device aging, and deployment-specific cooling setups—factors that become even more critical in chiplet-based designs because of the added complexity of interposer and package layers [74].

(ii) *Empirical identification approaches* instead learn compact lumped-parameter models directly from on-silicon measurements, often using synthetic workload excitations together with processor's architectural performance counters and on-die temperature sensors. These approaches naturally adapt to silicon variation and to the actual cooling environment, but most prior work targeted monolithic processors with a modest number of cores (up to 8–16). In such cases, dense excitation of all cores and simple geometric assumptions allowed fitting global MISO-ARX models and reconstructing the die floorplan from thermal couplings [26], [41].

Modern chiplet-based many-core processors break these assumptions. Cores are partitioned across multiple chiplets mounted on a common silicon interposer; thermal coupling is strong within each chiplet but weaker—yet still non-negligible—across chiplet boundaries [74], [120].

These limitations motivate the development of a new category of methods that employ distributed thermal-identification techniques to reconstruct the thermal model of chiplet-based architecture cores directly from measurement data [40], [94]. Such methods aim to provide compact, updatable models suitable for predictive thermal–power controllers in heterogeneous chiplet-based processors deployed in the field.

1.3 Motivation

From the above analysis, four key gaps emerge:

1. **Scalable PCS for many-core and chiplet-based designs:** The lack of an *open-source, research-grade* PCS architecture able to scale with the increasing number of cores and to sustain the sub-millisecond control loops required by advanced predictive policies.

2. **Characterization of power-management communication interfaces in HPC processors:** The growing complexity of power-management stacks, spanning operating systems, firmware, and low-level controllers, introduces non-negligible communication delays that can affect control responsiveness and energy efficiency. Despite the widespread adoption of delegation-based frameworks such as ACPI and SCMI, a quantitative understanding of their latency and its impact on runtime DVFS/thermal policies remains limited. A systematic, fine-grained characterization of these interfaces is required to assess their real-time behavior and to identify firmware or protocol-level optimizations that improve end-to-end power-management performance in modern HPC processors.

3. **Lack of a scalable control plane for distributed PCSs in chiplet-based processors:** As control functions become distributed across multiple chiplets, existing standards such as SCMI—designed for centralized agent–platform interaction—lack native support for low-latency, peer-to-peer coordination. Moreover, their strong dependence on kernel-space drivers hinders extensibility and experimentation. A lightweight and flexible communication substrate is required to decouple the control plane from the kernel and to enable scalable coordination among multiple PCSs within chiplet-based systems.

4. **Compact empirical thermal–power models for chiplet-based processors:** Simulation-driven models depend on proprietary floorplans and lose fidelity after aging or packaging variations, while existing empirical approaches have been validated only on small monolithic CPUs. New

identification techniques are required to capture thermal models directly from measurements on real silicon in chiplet-based architectures.

1.4 Contributions and Outline

This dissertation addresses the above gaps with the following contributions, each corresponding to one main chapter. Before the main contributions, chapter 2 introduces the physical and architectural foundations of power and thermal management in modern processors, covering transistor scaling limits, voltage–frequency regulation mechanisms, and basic control-state definitions.

1. **Chapter 3 - ControlPULP: an open, scalable PCS architecture.** We design and release *ControlPULP*, the first open-source, research-grade PCS built on a RISC-V parallel microcontroller cluster with DMA-accelerated sensor access and real-time OS support. Its parallel execution model allows scaling the computation of MIMO control laws with increasing core counts and enables the integration of advanced policies such as model-predictive thermal–power control.

2. **Chapter 4 - Performance Characterization of Power-Management Interfaces in HPC Processors.** Using an FPGA-accelerated HIL platform and a fully compliant Linux SCMI software stack, we provide the first fine-grained quantitative study of the communication delays in modern delegation-based power-management schemes. We measure message-dispatch, protocol, and FW processing overheads—e.g., 70.5 μs for dispatch and $\sim$ 603 μs for LLC response—showing how such delays propagate through the full control path and affect DVFS loop responsiveness. We further demonstrate that optimized FW can reduce the end-to-end command delay from $\sim$ 1.3 ms to $\sim$ 114 μs, yielding up to 3% shorter workload execution time.

3. **Chapter 5 - Towards User-Space Power-Management Communication Interfaces.** To support multi-chiplet packages with per-die PCSs, we implement a flexible control network built on the *Micro XRCE-DDS* middleware [44]. We introduce a new shared-memory transport layer that allows each PCS client to communicate efficiently with a central agent without kernel-mediated bottlenecks. The publish/subscribe abstraction of XRCE-DDS enables rapid addition of new data streams or PCS nodes and supports application-level custom power-management policies. We evaluate the infrastructure in terms of (i) Client-side processing latency per publication cycle, (ii) memory footprint, and (iii) message size on the *ControlPULP* PCS under different configurations, showing sub-100 μs processing latency and predictable memory usage below 2 KiB per topic.

4. **Chapter 6 - Compact Thermal and Power Models for Many-core Chiplet-Based Architectures.** We develop a data-driven methodology, tailored to many-core chiplet-based processors, that combines performance-counter-based per-core power estimation with distributed MISO-ARX identification of thermal couplings. The approach extracts inter-core thermal interactions directly from measurements on real silicon and reconstructs an

approximate processor floorplan to reduce model complexity with minimal loss in predictive accuracy. This yields compact yet accurate models achieving a per-core power RMSE of 2.464 W and a median thermal-prediction MSE of 3.756 $^{\circ}C^2$ under full-chip excitation on a 56-core Intel Sapphire Rapids processor.

Finally, the thesis is framed and concluded by two additional chapters:

- **Chapter 7 – Context and Technological Framework.** Positions the research within the broader European and international landscape of open, energy-efficient high-performance computing. It outlines the technological environment surrounding the development of the RISC-V–based *ControlPULP* subsystem and its integration within the SiPearl *Rhea1* processor under the *European Processor Initiative (EPI)*. The chapter also discusses the continuity of this work within the upcoming *DARE* project, which extends the presented methodologies toward distributed and dependable runtime management for heterogeneous HPC systems.
- **Chapter 8 – Conclusions and Future Work.** Summarizes the main achievements of this dissertation, discusses their implications for emerging chiplet-based and RISC-V architectures, and outlines potential directions for future research on open, distributed power and thermal management frameworks.

Overall, these contributions provide an open experimental platform and novel methodologies that jointly enable fine-grained, predictive power and thermal management in next-generation many-core and chiplet-based processors.

Chapter 2

Background & Fundamentals

2.0.1 Transistor Scaling and Power Consumption Considerations

The steady improvement in computing performance throughout the late 20th century was enabled by the simultaneous progress of lithography and transistor scaling. According to *Moore's Law* [79], the number of transistors per unit area doubles approximately every two years, whereas *Dennard scaling* [39] describes how device dimensions, voltage, and current should scale to keep power density constant as transistors shrink.

In the original Dennard model, all linear dimensions are reduced by a factor $k > 1$. To preserve the same electric field strength inside the device, the supply voltage V and the transistor threshold voltage V_T are also reduced by the same factor k, while the operating frequency f increases proportionally to k. Capacitance per device scales as $1/k$, and current as $1/k$, resulting in a power per transistor:

$$P = \alpha C V^2 f \quad \rightarrow \quad P' = \alpha \frac{C}{k} \left(\frac{V}{k} \right)^2 (kf) = \frac{\alpha C V^2 f}{k^2} = \frac{P}{k^2}. \tag{2.1}$$

At the same time, the number of transistors per unit area increases quadratically with the scaling factor k. If the original design contains N transistors over an area A, after scaling we have:

$$N' = k^2 N, \qquad A' = \frac{A}{k^2}. \tag{2.2}$$

Since the power per transistor scales as $P' = P/k^2$, the total power per unit area becomes:

$$\frac{P'}{A'} = \frac{N' P'}{A'} = \frac{(k^2 N)\,(P/k^2)}{A/k^2} = \frac{NP}{A} = \frac{P}{A}. \tag{2.3}$$

Therefore, even though transistor density increases by k^2, the overall power density remains approximately constant. This property defines the *constant-field scaling* regime, which allowed successive technology generations to achieve higher integration and frequency without exceeding thermal design limits. Each new technology node offered higher operating frequencies, reduced voltage, and lower energy per switching event, all while maintaining manageable thermal envelopes.

However, below the 90–65 nm technology nodes, the assumptions of constant-field scaling ceased to hold. The threshold voltage V_T cannot be reduced proportionally to V without causing excessive subthreshold leakage, which increases exponentially with reduced gate length and oxide thickness. As a result, voltage scaling stalled near $\sim$1 V, while the operating frequency continued to rise. This marked the transition toward *constant-voltage scaling*, in which feature size and switching speed continue to improve but voltage remains nearly fixed:

$$P' \approx \alpha C V^2 f' \propto f'. \tag{2.4}$$

Consequently, dynamic power scales linearly with frequency rather than remaining constant, and power density grows with each generation. In parallel, leakage currents I_{leak} contribute a static power component,

$$P_{\text{static}} = I_{\text{leak}} V, \tag{2.5}$$

which becomes increasingly dominant as billions of transistors remain biased even when idle. The combination of high dynamic power and rising static dissipation leads to *thermal density saturation*, often referred to as the *power wall*. Beyond this point, further increases in clock frequency result in disproportionate power and temperature rises, forcing designers to shift focus from frequency scaling to energy-efficient architectures and dynamic power management techniques.

2.0.2 From Transistor Physics to System-Level Power Management

As transistor scaling reached its physical and thermal limits, architectural and system-level mechanisms became essential to sustain performance improvements within a constrained power envelope. Modern processors integrate multiple techniques to dynamically regulate energy consumption and thermal dissipation according to workload conditions. The fundamental quantity governing dynamic power consumption in CMOS circuits is

$$P_{\text{dyn}} = \alpha C V^2 f, \tag{2.6}$$

where C is the effective switched capacitance, V the supply voltage, f the clock frequency, and α the activity factor representing the fraction of gates switching per cycle. Reducing either voltage or frequency lowers dynamic power quadratically or linearly, respectively. However, the switching speed of a transistor is limited by the effective overdrive voltage $(V - V_T)$, where V_T is the threshold voltage—i.e., the minimum gate-to-source voltage required to form a

conducting channel between source and drain. The propagation delay t_d of a CMOS gate is approximately proportional to

$$t_d \propto \frac{V}{(V - V_T)^\gamma}, \tag{2.7}$$

with γ typically in the range 1–2. As the supply voltage approaches V_T, the overdrive voltage decreases and transistor switching slows down sharply, limiting the maximum achievable clock frequency $f_{\max}$. This trade-off defines the fundamental constraint behind voltage–frequency scaling and motivates the use of dynamic power management schemes.

An Operating Performance Point (OPP) represents a valid pair (or tuple) of operating parameters—typically voltage and frequency (V_i, f_i)—that ensures stable and reliable circuit behavior within the device's electrical and thermal limits. Each OPP defines a safe region in the (V, f) space characterized by both timing and power constraints, and transitions between OPPs allow the system to balance performance and energy efficiency at runtime. OPPs can be defined for entire processors or for specific power domains such as cores, uncore, memory, or I/O. They are usually enumerated in firmware tables or device-tree descriptions and are accessible to the operating system or power-management framework through standardized interfaces such as ACPI or SCMI. Dynamic selection among OPPs enables *DVFS*, which jointly adjusts V and f according to workload activity, thermal headroom, or global power budgets.

The concept of OPP is specialized in many commercial architectures through the definition of *performance states* (P-states). In this model, each processor exposes a discrete set of frequency–voltage operating points, ordered from P0 (maximum performance, highest power) to progressively lower states (P1, P2, ...) offering reduced performance and energy consumption. Transitions between P-states are commanded by the OS or a runtime power manager, which issues performance requests to the underlying power controller through standardized protocols. The controller uses on-chip PVT sensors to monitor temperature and voltage conditions in real time. Based on these readings, it programs the PLLs and VRMs to achieve the closest safe operating point in the predefined OPP table. Although modern control interfaces (e.g., ACPI, SCMI, or RPMI) allow asynchronous requests, the combined latency of firmware processing and voltage settling remains in the order of tens to hundreds of microseconds, introducing a lower bound on the achievable control-loop period.

Complementary to performance states are *idle states* (C-states), which specify progressively deeper levels of inactivity when a core or subsystem has no instructions to execute. In C0 the core is fully active; in C1 the pipeline is halted while clocks remain enabled; deeper states such as C2 or C3 successively disable caches, clock trees, or entire power domains to minimize leakage. Each transition to a deeper idle state reduces static power consumption at the cost of increased wake-up latency. Operating systems and firmware employ heuristics or predictive models to select appropriate C-states based on workload behavior, balancing energy savings and responsiveness.

Chapter 3

ControlPULP: A RISC-V On-Chip Parallel Power Controller for HPC

3.1 Introduction

After the end of Dennard's scaling, the increase in power density has become an undesired but unavoidable collateral effect of the performance gain obtained with integrated systems' technological scaling. An increase in power density has multiple adverse effects, which are collectively referred to as the *power wall*: components' lifetime shortening, electromigration, dielectric breakdown due to thermal hot spots and sharp thermal gradients, and degraded operating speed due to leakage current exponentially increasing with temperature. This trend has made the processing elements (PEs) at the heart of computing nodes energy, power, and thermally constrained [66].

Two approaches have been adopted to mitigate the *power wall* at the system level [116]: static thermal management (STM) and dynamic thermal management (DTM) techniques. The former allows increasing the TDP sustained by the chip with a tailored design of heat sinks, fans, and liquid cooling. However, STM strategies incur increasingly unsustainable costs when over-designed to remove heat in the worst-case conditions for today's HPC processors. Consequently, DTM techniques have become more and more crucial to bound the operating temperature with *run-time active control*, for example, by exploiting PVT sensors along with DVFS, thread migration/scheduling, throttling, and clock gating. Hence, standard cooling systems can be designed to handle the average case, leaving the management of peaks to active control.

Modern high-performance processors feature many cores integrated into a single silicon die. Recent notable examples are AWS Graviton 3 (64 Arm Neoverse V1 cores) [64], Intel Raptor Lake (24 cores, 32 threads) [57], AMD

Epyc 7004 Genoa (up to 96 Zen 4 cores) [4], SiPearl Rhea Processor (72 Arm Neoverse V1 Zeus cores) [48], Ampere Altra Max (128 Arm Neoverse N1), and the NVIDIA Grace CPU (144 Arm Neoverse V2 cores). Their application workload requires runtime dynamic trade-off between maximum performance (fastest operating point [33]) in CPU-bound execution phases and energy efficiency in memory-bound execution phases (energy-aware CPU [105]).

While software-centric advanced DTM policies have been proposed [23, 25, 116], they mainly execute on the CPU's application-class processors (APs), playing the role of HLCs governors. Nevertheless, in recent years it has become clear the trend of abstracting power and system management tasks away from the APs [1] [73] towards control systems that are closer to the controlled hardware components and can guarantee faster, and more precise control actions, namely LLCs.

Modern processors integrate on-die LLCs [72] in the *uncore* domain, referred to as PCSs, as dedicated embedded hardware resources, co-designed with a PCF implementing complex MIMO power management policies. Advanced DTM involves embedding and interleaving a plurality of activities in the PCS, namely (i) dynamic control of the CPU power consumption with short time constants [98], required to prevent thermal hazards and to meet the TDP limit (power capping [70]), (ii) real-time interaction with commands provided by on-die (OS - power management interfaces and on-chip sensors) and off-die (BMC, VRMs) units and (iii) dynamic power budget allocation between general-purpose (CPUs) and other integrated subsystems, such as GPUs [105].

Existing on-die PCSs share a similar design structure. They feature an embedded single-core MCU [1] supported by dedicated hardware state machines [105] or more generic accelerators [102]. The hardware typically takes advantage of specific software libraries [2] to implement the real-time execution environment required to run power management policies under tight timing constraints. Many-core power management demands fine-grained control of the operating points of the PEs [72] to meet a given processor power budget while minimizing performance penalties. The control policy has to provide fast and predictable responses to promptly handle the incoming requests from the OS or BMC and prevent thermal hazards. A flexible and scalable way to sustain these computationally intensive operations is required to provide accurate control per core and to support more advanced control policies, such as those based on model-predictive control [23].

Furthermore, simultaneous HW and SW development of PCS and PCF has to be coupled with a dedicated co-design and validation framework within the *controlled system*. Indeed, PCS and PCF performance directly depends on their interaction with the physical state of the controlled processors — temperature, power, workload, and control decisions from the HLCs. Such a physical state time-scale model requires near real-time speed to be meaningful. This involves, on the one hand, the design of adequate on-chip interfaces between the controller

[1] `https://github.com/Arm-software/SCP-firmware`
[2] `https://github.com/open-power`

to be validated — integrated system under test (ISUT) — and the surrounding system. On the other, a proper *virtual* representation of the system surrounding the ISUT that models physical components on a target computer [107], called *plant model*. In a design relying on dedicated PCSs, the controlled plant is a complex MIMO multi-processor system on chip (MPSoC) with APs that is often not available during the design phase of the PCS to be integrated, thus being replaced by a thermal and power model encapsulating floorplan, power and thermal information of the CPU under a particular application workload.

Today's advanced heterogeneous system on chip (HeSoC) integrate both single/multiprocessors MCUs and configurable hardware components on the same die (FPGA-SoC). Their native integrated and configurable structure makes them ideal options for closed-loop emulation of on-chip PCSs targeting advanced power management.

Therefore, the central idea of the work presented in this chapter is to fulfill a twofold need: on the one hand, the design of a capable PCS architecture optimized for handling a per-core, reactive thermal and power control strategy within the required power budget and timing deadlines. On the other, the design of an agile, closed-loop framework for the joint HW/SW development of integrated control systems, leveraging the capabilities of modern HeSoC platforms.

To the best of the authors' knowledge, this work presents the first research platform where the full-stack (HW, real-time OS (RTOS), PCF, and power/thermal emulation setup) required for LLC-driven on-chip power and thermal management is released open-source [3]. Unlike traditional Linux-capable symmetric multiprocessing (SMP) multi-core systems, as a research platform for on-chip power and thermal management, ControlPULP aims to keep the design's HW and SW complexity to a 32-bit manager and 32-bit programmable many-core accelerator (PMCA) with RTOS support. This design choice better fits the resource constraints of a controller embedded in the *uncore* domain of a larger CPU that is assumed to integrate out-of-order, massive application-class processing elements already.

This chapter presents the contributions and results reported in the journal publication [88], which provides an extended version of the research proposed in [87]. The main contributions presented in this chapter are summarized as follows:

1. **End-to-end RISC-V parallel PCS architecture.** We design an open-source, fully programmable PCS architecture named *ControlPULP*, based on open RISC-V cores and hardware IPs [103]. ControlPULP represents the first fully open-source PCS with a configurable number of cores and hardware resources, scalable to match the computational requirements of increasingly complex power-management policies in current and future high-performance processors, as well as the first proposed in the RISC-V community. The architecture integrates a *manager core* coupled with a

[3] `https://github.com/pulp-platform/control-pulp`

multi-core PMCA (cluster) featuring per-core FPUs for reactive control computation. The cluster subsystem includes a DMA engine to accelerate data transfers from on-chip sensors, supporting 2D stride patterns essential for reading from PVT sensors with equally spaced address mappings (Sec. 3.5.3).

2. **Real-time optimization for power management.** We tailor ControlPULP to meet real-time constraints typical of embedded power management tasks. The architecture incorporates a fast Core Local Interrupt Controller (RISC-V CLIC [99]) to process high-frequency interrupt messages associated with OS- and BMC-driven commands, and a low-latency, predictable interconnect infrastructure to ensure deterministic control-loop timing (Sec. 3.5.3).

3. **Closed-loop evaluation framework.** We demonstrate the end-to-end capabilities of ControlPULP through a case study focused on the quality of service (QoS) control algorithm. A closed-loop evaluation framework is developed following the concept of HeSoCs, enabling real-time characterization of the control policy while fast-prototyping the underlying hardware. The framework combines a power, thermal, and performance model of the controlled CPU—treated as the *plant*—with workload instruction traces and the PCF to form a complete feedback loop. This setup allows multi-level validation and characterization across software in the loop (SIL), MIL, and HIL abstraction layers (Sec. 3.5.5).

4. **Benchmarking against state-of-the-art controllers.** We benchmark the reactive control algorithm implemented in ControlPULP against IBM's OpenPower framework—one of the few publicly available, industry-grade control policies. Under identical conditions, ControlPULP achieves approximately 5% higher precision in set-point tracking in MIL simulations, highlighting the benefits of fine-grained, multi-core PCS architectures for thermal and power control (Sec. 3.5.5).

3.2 Related Work

There is little publicly available information on PCSs architectures and their HW/SW interface. Table 3.1 summarizes the leading solutions from academia and industry.

Intel's PCU, introduced with Nehalem microprocessor [51], is a combination of dedicated hardware state machines and an integrated microcontroller [105]. It provides power management functionalities such as DVFS through voltage-frequency control states (P-states and C-states), selected by the HW (*Hardware-Managed P-States*). The PCU communicates with the PEs with a specialized link through *power management agents*. Intel's main control loop runs at $500\mu s$ [108].

AMD adopts a multiple power controller design, with one SMU for each CPU tile (group of cores) in a Zeppelin module. All SMUs act as slave components,

Table 3.1: Comparison among existing proprietary and freely-available PCS from industry and academia.

pcs	Provider	HW	pcf scheduling	Programmable accelerator	Openness (HW/SW)
Industry					
PCU	Intel	1-core, HW FSMs	n.a.	✗	✗ ✗
SCP, MCP	Arm	2-cores	SW FSMs	✗	✗ ✓
SMU	AMD	1-core	n.a.	✗	✗ ✗
OCC	IBM	1-core, microcode engines	SW FSMs	✗	✗ ✓
Academia					
Bambini et. al [17]	academic	1-core	RTOS	✗	✗ ✓
ControlPULP	academic	1-core, cluster accelerator	RTOS	✓	✓ ✓

monitoring local conditions and capturing data. One of the SMUs also acts as a master, gathering all information from the slave components and then choosing the operating point for each core [31].

Arm implements two independent PCSs based on the Arm Cortex-M7 microcontroller, SCP and MCP. The SCP provides power management functionality, while the MCP supports communications functionality. In Arm-based SoCs the interaction with the OS is handled by the SCMI protocol [12]. SCMI provides a set of OS-agnostic standard SW and HW interfaces for power domain, voltage, clock, and sensor management through a shared, interrupt-driven mailbox system with the PCS.

The IBM OCC, introduced with Power8 microprocessor, is composed of 5 units: a central PowerPC 405 processor with 768 KiB of dedicated SRAM and four microcode general-purpose engines (GPEs). The latter are responsible for data collection from PVT sensors, performance state control and CPU stop functions control (*PGPE* and *SGPE*) respectively. IBM OCC's PCF is called OpenPOWER, and has a periodicity of 250μs [102]. It relies on a *frequency voting box* mechanism to select a frequency for each core conservatively based on the minimum input - highest *Pstate* - from several independent power-control (*control vote*) and temperature-capping (*thermal control vote*) features. The *thermal control vote* consists of one PID that reduces the frequency of each core based on the temperature of the hottest PE. Furthermore, similarly to Arm's SCMI standard, IBM's OCC relies on a *command write attention/interrupt mechanism* to notify the *PGPE* of an incoming asynchronous command/request to be processed[2], such as the desired *PState*. *PGPE* arbitrates this information with the voting box output from the PowerPC 405 according to a minimum *PState* policy.

Last, Bambini et al. [17] show that a single-core, RISC-V-based microcontroller can execute similar reactive control algorithms with a control loop of 500μs. The work relies on the SPI interface to conduct the main periodic task and lacks support for essential HW IPs such as DMA and floating point unit

(FPU).

All the state-of-the-art (SoA) power controllers lack the flexibility and scalability of a multi-core architecture supported by adequate I/O bandwidth from/to on-die and off-die power monitors and actuators coupled with fast interrupt handling hardware for HLCs (OS and BMC) commands dispatching, which is the critical innovation provided by ControlPULP.

It is essential to notice that PCSs design is only half of the coin, as its performance depends on the PCF and real-time performance achieved in closed-loop. Several works have targeted the emulation of 'in-field' power management algorithms, but none of them has validated the PCS and PCF designs jointly (co-designed).

Atienza et al. [13, 14] propose a thermal emulation framework where a generic MPSoC is implemented on FPGA. A host computer executes the *thermal model*, which is driven by real-time statistics from processing cores, memories, and interconnection systems emulated on the FPGA. With the increasing number of cores in modern CPUs, an FPGA approach that implements the entire MPSoC is not feasible and incurs resource partition with high platform costs. Beneventi et al. [25] design a similar closed-loop approach where a subset of the Intel Single-Chip-Cloud computer's PEs execute the *thermal model* while receiving online workload information from the remaining PEs. While employing actual and commercial hardware in the emulation, the work focuses on the HLC only and with a software-centric methodology, being HW/SW co-design of the LLC prevented by the inaccessibility of the underlying HW.

Beyond these approaches, our work aims at co-designing the LLC (PCS) and HW/SW components of the controlled plant to assess "in-field" performance. This step is well understood in the design flow of classic control systems targeting automotive, avionics, and robotics domains, where progressively more realistic simulations of the plant in the closed-loop are coupled with the introduction of the actual hardware controller (HIL) and checked against model-based closed-loops (SIL, MIL) from early development stages of the control design.

A taxonomy of the various design possibilities for a control system that adapts well to the scenario introduced in this chapter is provided in [30]. On the controller side, an FPGA emulation approach provides the benefit of testing the control firmware developed during SIL/MIL on the actual hardware controller with the guarantee of cycle-accurate simulation. The latter is required to achieve fine-grained hardware observability and controllability [107] and one-to-one correspondence between the register transfer level (RTL) source and its FPGA mapping in terms of clock cycles (Direct FPGA Architecture Model Execution (FAME) systems [114]). On the plant side, a virtual simulation of the plant is preferred to an FPGA-based approach, which incurs high development costs due to the complexity of the whole plant system to be emulated. For this purpose, MCUs are the solution adopted by the industry nowadays: they are cheap, general-purpose and with a standardized and automated software development process.

The combination of MCUs and FPGAs flexibility takes the best of both worlds and is supported by modern, commercial off-the-shelfs (COTSs) FPGA-

SoCs platforms such as Xilinx Zynq Ultrascale+ and Versal families, making it the solution of choice adopted in our work.

3.3 The ControlPULP platform

ControlPULP extends the single-core architecture of commercial controllers, introducing the first multi-core RISC-V PCS architecture. For completeness, the following sections first present a high-level overview of a generic HPC CPU integrating the PCS. The platform's software stack is then described to clarify the control-policy flow, the interaction between the controller and the controlled plant, and the terminology used throughout this chapter (Sec. 3.3.1). Finally, the hardware architecture and design trade-offs of ControlPULP are discussed (Sec. 3.3.2) in relation to the implementation of the control algorithm.

3.3.1 Controlled plant and Power Control Firmware

Fig. 3.1 depicts the high-level structure of the HPC CPU silicon die. From a control perspective, we distinguish between the controlled system (*plant*) and the on-chip LLC controller, namely ControlPULP. Furthermore, the figure illustrates the environment surrounding the CPU socket hosting the silicon die, namely the motherboard, with off-chip actors, such as VRMs and BMC. The OS running on the PEs, as well as the off-chip BMC, are the two HLCs.

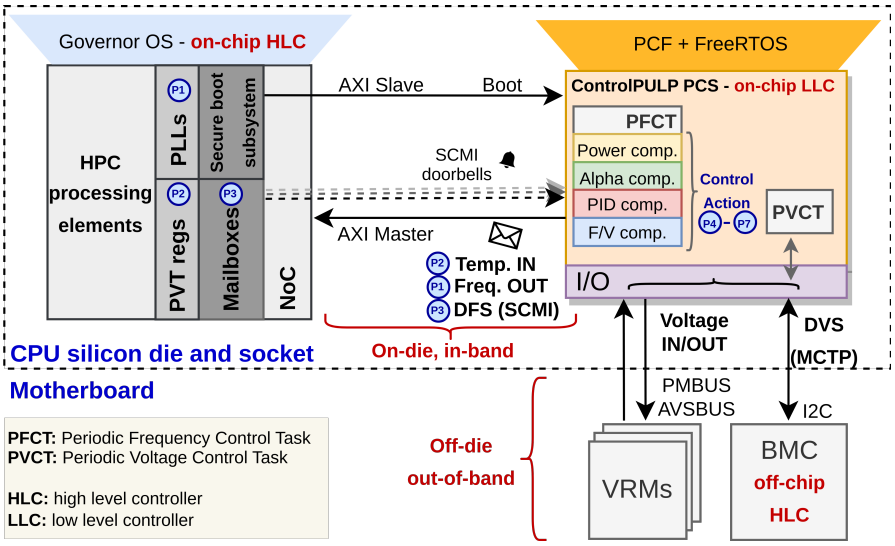


Figure 3.1: High-level overview of the system. We highlight on-chip and off-chip HLCs (OS and BMC), the LLC (ControlPULP) and the MIMO IO interfaces. Furthermore, the figure details the PCF phases described in Sec. 3.3.1

We assume the controlled plant is a many-core HPC CPU with 72 application-class PEs, and exposes hardware mailboxes through a network on chips (NoC) interconnect system. While the mailboxes mediate the dynamic frequency scaling (DFS) commands (e.g. target frequency) dispatched on-chip by the governor OS HLC to the LLC, power management protocols such as Power Management Bus (PMBUS), Adaptive Voltage Scaling (AVSBUS) and Management Component Transport Protocol (MCTP) mediate dynamic voltage scaling (DVS) commands (e.g. power budget threshold) from the BMC HLC, as detailed in Sec. 3.3.2.

To clarify the terminology in Fig. 3.1, *on-die* designates any element of the HPC CPU that resides on the chip die, such as PVT sensors and registers, frequency actuators, and mailboxes. *In-band* services refer to SCMI-based interaction and PVT data acquisition. Lastly, *off-die* indicates VRMs communication and BMC requests through *out-band* services.

The PCF executes the thermal and power control functions and manages on-die and off-die communications and data transfers. Real-time priority-driven scheduling with static task priorities and preemption is required to manage the control functions. In this work, we use FreeRTOS, an industry-grade, lightweight, and open-source operating system for microcontrollers. The software stack of the proposed platform is shown in Fig. 3.2.

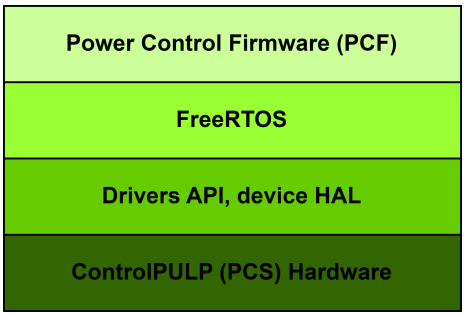


Figure 3.2: ControlPULP software stack. The application control policy (PCF) executes on top of FreeRTOS, which controls the hardware with target-specific drivers and HAL APIs

The PCF routine implements a two-layer control strategy [17], managed by the FreeRTOS scheduler as two periodic tasks characterized by multiple harmonic frequencies: the **PFCT** — 2 kHz, i.e. $500\mu s$ — and the **periodic voltage control task (PVCT)** — 8 kHz, i.e. $125\mu s$. Splitting the control routine into two tasks grants more fine-grained control actions and helps meet different performance requirements and sensors-update frequencies. For instance, power changes rapidly due to instruction-level variation of the effective switching capacitance of the computing units, while temperature variations are slower. The control policy has to handle these widely split time scales. Furthermore, VRMs generally update more frequently than temperature sensors.

The PFCT is the main control task. It receives the desired clock frequency

operating point for each processing element from the OS HLC governor as well as a power budget threshold from the BMC HLC via the PVCT, and computes the optimal frequency/voltage pair to meet the physical and imposed constraints of the system. The task is then responsible for dispatching the controlled frequency by directly accessing the CPU PLLs, as from Fig. 3.1. The PFCT executes a two-layer control strategy [17] consisting of a *power dispatching layer* and a *thermal regulator layer*. PFCT's control step n comprises several phases, illustrated in Fig. 3.1: **(P1)** allocate the controlled clock frequency computed at step $n-1$ to each core; **(P2)** read the PVT sensor's registers and the workload characteristics from each core; **(P3)** obtain commands and information on the constraints (DVFS operating points, power budget) from the OS and the BMC; **(P4)** compute the estimated power for each core and the total consumed power of the system; **(P5)** apply a power capping algorithm, such as *alpha* [17] when the total power exceeds the power budget constraint; **(P6)** further reduce the power of each core through PIDs computation when the temperature at phase (P2) exceeds the threshold; **(P7)** compute both the frequency and voltage to be applied to the controlled system. Throughout this manuscript, we name *control action* the computational body of the PCF execution (P4)-(P7).

The PCF does not provide per-core voltage control but groups PEs in coarse-grained voltage domains. The PVCT is responsible for detecting the changes in the system's power consumption. It periodically reads the power consumption of the voltage rails — associated with each voltage domain — from the VRMs and programs micro-architectural power/instruction throughput capping interfaces (if supported by the PEs). Lastly, it modifies the PFCT's power budget threshold as requested by the BMC. Even though the PFCT computes both optimal frequency and voltage, it only dispatches the frequencies to apply at phase (P1) in step $n+1$. In contrast, the PVCT dispatches the voltages at step n (one iteration before), hence the names chosen for the two tasks.

3.3.2 System architecture

Fig. 3.3 provides a block diagram overview of ControlPULP. The top-level subsystem of the design is the *manager domain*, consisting of a CV32E40P open-source [4] industry-grade processor core and a set of system I/O interfaces to interact with external peripherals and memory-mapped devices (Sec. 3.3.2). The primary micro-controller-like subsystem is also a recurrent element in the SoA designs surveyed in Sec. 3.2.

Real-time and predictability

In the following, we discuss the architectural design decisions concerning RAM banking and interrupt processing taken to make the design more suitable for real-time workloads.

[4] `https://github.com/openhwgroup/cv32e40p`

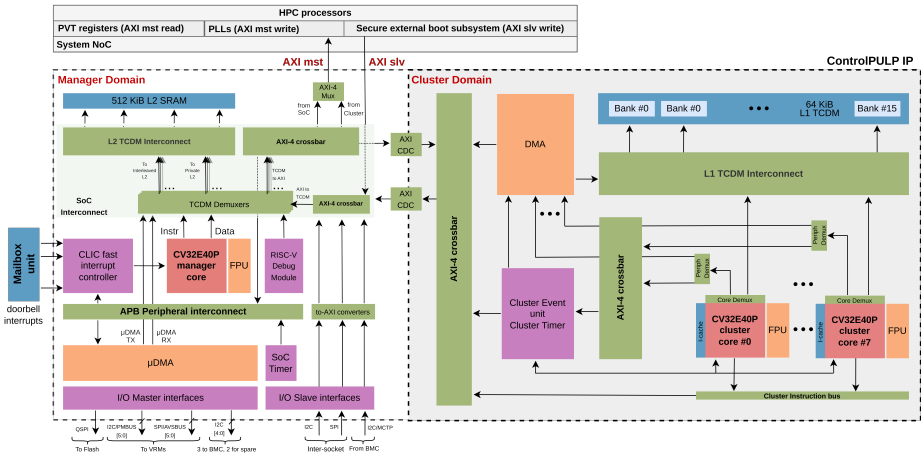


Figure 3.3: ControlPULP hardware architecture. On the left, the *manager domain* with the *manager core* and surrounding peripherals. On the right, the *cluster domain* accelerator with the eight cores (workers)

L2 memory banks constant access time The L2 RAM, which is the RAM block connected to the *manager domain* and system I/O interfaces, is sized to 512 KiB, enough to fit the whole firmware binary and data so that no swapping is required. The L2 RAM comprises six banks. The access time to each bank is constant when there are no access conflicts. Two of these banks are marked private to prevent DMA transfers from peripherals and other components from disturbing the manager core's instruction and data fetching. The manager core has exclusive access to those.

Low constant latency core-local (CLIC) interrupt controller Provided the need for a real-time and predictable architecture, we extend the original core-local interruptor (CLINT), compliant with RISC-V privileged specifications [115], with the newly proposed core-local interrupt controller (CLIC), currently under ratification by the RISC-V community. We employ an open-source implementation of the CLIC [5] [16] that reflects the latest status of the RISC-V CLIC draft specifications [99]. The integration process includes the addition of Control and Status Registers (CSRs)CSRs registers in the processor's micro-architecture as per specifications.

The CLIC introduces several improvements to the standard CLINT to achieve faster interrupt handling. Among those are dedicated memory-mapped registers for software configurable interrupt priority and levels at the granularity of each interrupt line, runtime-configurable interrupt mode and trigger type, and support for interrupt preemption within the same privilege level (interrupt nesting). Selective hardware vectoring (SHV) enables the programmer to optimize each

[5] `https://github.com/pulp-platform/clic`

incoming interrupt for either faster response (*vectored* mode, when each interrupt service routine has a reserved entry in the interrupt table) or smaller code size (*direct* mode, when each interrupt traps to the same exception handler address). Lastly, with the CLIC, local interrupts can be extended to 4096 lines instead of being limited to the processor's XLEN (32-bit for CV32E40P).

In the work reported in this chapter, we implement 256 local interrupt lines coming to ControlPULP from the mailbox infrastructure (Sec. 3.3.2). The CLIC configuration helps reduce the interrupt response latency and is capable of entering the interrupt handler within 30 clock cycles (Sec. 3.5). This is a crucial property to increase responsiveness on external, agent-driven requests.

Cluster accelerator

To meet the computational demands of the control algorithms, in particular, when scaling to a large number of controlled high-performance PEs and improving the control performance, we opt for a flexible programmable accelerator, namely a cluster of RISC-V CV32E40P cores — referred to as *workers* in this manuscript — tightly coupled to 64 KiB RAM (L1) and a DMA engine. The accelerator is represented in Fig. 3.3 as *cluster domain*.

Multi-core computing system Control algorithms (Sec. 3.3.1) can be parallelized on the cluster domain (Sec. 3.5.2). This guarantees a high grade of flexibility on the software development side, and is in sharp contrast with hard-wired control logic featured in SoA controllers (Sec. 3.2), which lack flexibility. The *manager core* offloads the control algorithm to the team of workers in the cluster. Each worker features a private instruction for the instructions stored in the main L2 memory and accesses L1 through a single-cycle latency logarithmic interconnect.

In the most straightforward parallelization scheme, a worker computes the control action (Sec. 3.3.1) for a subset of the controlled cores. The number of workers in the cluster is parametric. In the following, we consider eight cores to demonstrate scalability. Each core in the cluster features an FPU with a configurable number of pipeline stages. In our instantiation, we use one internal pipeline stage, which is sufficient to meet our frequency target (Sec. 3.5.1). Furthermore, Montagna et al. [78] show that this configuration achieves high performance and reasonable area/energy efficiency on many benchmarks.

2-D DMA transfer engine The *cluster domain* integrates a multi-channel DMA with direct access to L1 RAM and low-programming latency (62 clock cycles, Sec. 3.5.3). The DMA's main task is to provide direct communication between L2 and L1 memories in parallel and without intervention from the *manager* or *cluster* domains [104].

We tailor the DMA's capabilities to suit the control policy use case by (i) directly routing the cluster DMA to the PVT sensors registers through the outgoing AXI master interface, which guarantees flexibility by decoupling data transfers and computation phases, (ii) exploiting 2-D transfers for equally spaced

PVT registers accesses and (iii) increasing the number of outstanding transactions (up to 128) to hide the latency of regular transfers.

Commercial PCSs described in Sec. 3.2 also decouple the actual computation from data acquisition. For instance, according to available SoA information, IBM's OCC employs general-purpose cores (GPEs) tasked to read PEs's data and temperatures instead of a dedicated data mover engine with reduced programming interface overhead [102].

System I/O interfaces

AXI4 interfaces ControlPULP features two AXI4 ports, one *master* and one *slave*, with 64-bit W/R, 32-bit AW/AR wide channels. They play a crucial role in the design and guarantee low-latency communication with the controlled system. The AXI slave drives the booting process of the PCS. In the high-level structure depicted in Fig. 3.1, an external, secure subsystem is responsible for loading the PCF binary into ControlPULP's L2 SRAM by mapping to a region of the L2 SRAM. The AXI master is the transport layer over which the PCS collects PVT sensors data and power policy target requirements. On the other hand, it dispatches the computed optimal operating point to the PEs during the control policy (Sec. 3.3.1). Its address space is reachable from both the *manager* and *cluster* domains.

Mailbox-based SCMI communication ControlPULP adopts and implements the Arm standard SCMI protocol to handle external power, performance, and system management requests from the OS HLC. SCMI allows an OS kernel that supports SCMI to interact with ControlPULP without requiring a bespoke driver. Furthermore, the design of the SCMI protocol reflects the industry trend of delegating power and performance to a dedicated subsystem [73]. SCMI involves an interface channel for secure and non-secure communication between a caller (named *agent*, i.e., one processing element of an HPC CPU) and a callee (named *platform*, i.e., ControlPULP). The latter interprets the messages delivered by the former in a shared memory area (mailbox region, Fig. 3.1) and responds according to a specific protocol. The proposed PCS implements an interrupt-driven transport mechanism through the CLIC. In our use case with 72 controlled cores, the platform can process up to 144 secure, and non-secure interrupt notifications.

We design hardware mailboxes as the transport layers for the SCMI communication mechanism. The shared memory region is implemented according to the single-channel memory layout described in the specifications. We reserve a space of 8B for the implementation-dependent payload field, totaling 40B per channel. Each channel is associated with one outgoing interrupt line in the agent-platform direction (doorbell). The agent identifier is encoded with the message such that more agents can use the same channel as the notification vehicle. Hence, the number of interrupts dispatched by the mailbox system can be smaller than the number of agents, with the benefit of reducing the area of the interrupt controller's configuration register file. The agent identifier is not described in

the official specifications and is mapped to a reserved field of the single-channel memory layout. With 64 channels as employed in the proposed implementation, the shared memory region has a size of about 2kiB.

I/O peripherals for voltage rails power management ControlPULP integrates a peripheral subsystem in the *manager domain*, where an I/O data engine unit (named μDMA IP) enables autonomous communication between off-die elements and the L2 SRAM with low programming overhead for the manager core. In the presented design, we upgrade the peripheral subsystem with industry-standard power management interfaces to handle off-die communication services. The PCS integrates 6 AVSBUS and PMBUS interfaces towards VRMs. The PMBUS and AVSBUS bus protocols extend I2C and SPI to monitor voltage and power rails digitally, preserving optimal speed/power consumption trade-off. 5 I2C master/slave interfaces manage the communication with the BMC and other board-level controllers. The slave interfaces transfer DVS operating points (power budget) from the BMC according to the Platform Level Data Model (PLDM) and MCTP transport layer protocols.

3.4 FPGA-based HIL thermal, power, performance and monitoring emulation framework

This section provides an end-to-end description of the HeSoC based HIL emulation methodology. The closed-loop approach that we implement introduces the actual LLC executing the PCF as ISUT and relies on a *thermal, power and performance model* for the controlled CPU plant. While the control literature refers to such a setup as processor in the loop (PIL) or FPGA in the loop (FIL), others already define a closed-loop as HIL when the connection between the integrated system under test and the plant reflects the actual hardware interface of the final manufactured silicon, without relying on a virtual representation [107]. Since this is the case for the present work, we adopt the HIL terminology.

3.4.1 HIL system and PCS mapping on FPGA

The Xilinx Ultrascale+ FPGA family is widely adopted in the heterogeneous computing domain [63]. It features a Processing System (PS), or host computer, and a Programmable Logic (PL), namely the configurable FPGA, integrated on the same physical die. Fig. 3.4 provides an overview of the FPGA based emulation framework designed in this work and its main actors.

The PS consists of an industry-standard, application class, quad-core, 64-bit Armv8® Cortex-A53 *Application Processing Unit* with 32 KiB L1 instruction and data cache per core and a 1 MiB L2 cache shared by all four cores — on-chip memory (OCM) —, clocked at 1.2 GHz, a dual-core Cortex-R5F *Real Time Unit*, and Mali™-400 MP2 GPU based on Xilinx's 16nm FinFET.

We implement ControlPULP on the ZCU102 PL with Xilinx Vivado 2022.1. The PS interfaces with ControlPULP through the AXI4 master and slave ports

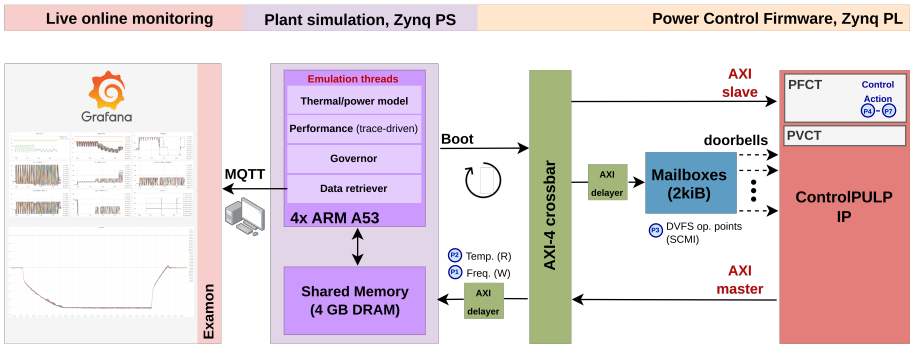


Figure 3.4: HIL test procedure on an FPGA-SoC with ControlPULP

(Sec. 3.3) and provides it with an external reset and 125 MHz clock, which is internally converted to 50 MHz system clock by the ISUT. Table 3.2 shows the board's resource utilization inferred by the implementation. Overall, ControlPULP fits the available resources with almost 97% utilization.

Table 3.2: ControlPULP resource utilization on the Xilinx UltraScale+ ZCU102.

Resource	Utilization	Available	Utilization [%]
LUT	265718	274080	**96.95**
FF	153155	548160	27.94
BRAM	278	912	30.48
DSP	93	2520	3.69
I/O	83	328	25.30

The communication between the plant simulation and the ISUT is regulated through a 4 GiB off-chip DDR4 DRAM provided on the board, which allows them to exchange data and realize the feedback loop with up to 19.2 GB/s of bandwidth.

3.4.2 Plant simulation

In the considered control scenario, the plant provides a thermal, power, performance, and monitoring framework capable of simulating the power consumption and temperature of a high-end CPU processor. The plant simulation is programmed in *embedded C*. Its functionalities are split into several threads employing `pthread` Linux libraries to exploit multi-core parallelism. In the following, we refer to Fig. 3.4 and detail the threads' organization and interaction with the ISUT.

The ***thermal and power model thread*** is the leading simulation thread. It carries out the computation of the thermal and power model with a periodicity of $1\mu s$ which, according to the literature [24, 18], is fast enough to capture the thermal dynamics, i.e., three orders of magnitude faster than the fastest thermal time constant $\sim 1ms$, and to simulate power spikes and oscillations that are not filtered by the hardware power delivery network (up to $\sim 100\mu s$ [38]). The thread takes as input the values controlled and dispatched by the PCS controller (frequency and voltage) and the modeled workload from the *performance model thread* to compute the average consumed power and the temperature of each core. The average consumed power is computed with an algebraic model of the cores, which includes core-to-core variability and noise, according to [18]:

$$P_i = P_{i,\text{static}} + P_{i,\text{dynamic}} = \kappa(T_{\text{Si},i}) \cdot (I_{\text{cc},i} \cdot V_{\text{dd}} + (C_{\text{eff},i} \cdot f_i \cdot V_{\text{dd}}^2)) \qquad (3.1)$$

where κ represents the temperature dependency of the computed power, and C_{eff} is the equivalent effective capacitance of the controlled CPU. The temperature is simulated through a discrete state space model, which considers the temperature and instantaneous power of the neighboring simulated cores. Coefficients are extracted from a commercial multi-node RISC-V cluster capable of providing an HPC production stack, Monte Cimone [22].

The ***performance model thread*** assists the *thermal and power model thread* by providing the workload characteristics for the next interval of time. We rely on a *trace-driven* approach for the simulated workload for practical reasons: (i) the traces could be extracted from accurate benchmarks such as PARSEC [28], SPEC [81] or real HPC applications; (ii) simpler but effective performance models can be built on top of workload traces, e.g., instructions per cycle (IPC) model and roofline model [54], enabling the evaluation of the impact of the power management policies; (iii) the power consumption is mainly affected by workload composition, i.e., memory bandwidth and vector/single instruction, multiple data (SIMD) arithmetic density.

In this work, we craft a synthetic benchmark (`Wsynth`) that stresses the control corner cases and consists of maximum power (`WsynthMax`) and idle power (`WsynthIdle`) instructions, mixed power instructions (`WsynthMix`) and lastly instructions with different power densities and fast switching (`WsynthFast`) to stress the power limiter and the shorter timing constants of the temperature response.

The ***governor thread*** emulates command dispatching agents such as the operating system running on the CPU or the off-chip BMC in the motherboard. They are tasked to send requirement directives to the controller, such as DFS and DVS operating points. In the plant simulation presented in this manuscript, the OS and BMC communication is modeled through the SCMI transport layer. Future work will integrate PLDM/MCTP transport layers supported by ControlPULP (sec. 3.3.2) in the HIL framework for BMC-related communication.

Finally, the ***data retriever thread*** collects all the simulation data and periodically dispatches them through the network. It relies on Eclipse Mosquitto

as a message broker, which implements the Message Queuing Telemetry Transport (MQTT) network-based messaging protocol. Collected data are fed to *Examon* [29], an open-source [6] framework that enables performance and energy monitoring of HPC systems. *Examon* relies on *Grafana* as interactive data-visualization platform with live-updated dashboards.

3.4.3 HIL testing procedure

We identify two main phases to carry out the emulation. In the *system setup* phase, the communication between PS and PL happens in a conductor-follower fashion. The PS drives the PCS deployment on the FPGA, its booting process, and firmware binary flashing. The AMBA AXI4 protocol regulates the data transmission across this communication channel. To this aim, we generate a complete embedded SMP Linux system paired with a persistent root file system on top of the four Arm A53 processors with Buildroot [7].

In the subsequent *system emulation* phase, PS and PL execute the respective routines and communicate *asynchronously* with each other through the shared DRAM memory region. The lack of an explicit synchronization point between the controller and the simulated plant is inherent to the nature of the control since dynamic thermal and power management involves run-time active control. At the same time, the underlying MPSoC varies its workload to meet a specific computational need.

The *system's setup and emulation* phases consist of the following steps:

1. Linux First Stage BootLoader (FSBL): the PL is programmed with ControlPULP's *bitfile*, which contains the hardware design information of the controller.

2. Linux *U-Boot*: Linux kernel boots on the Arm Application Processing Unit (APU) cores.

3. ControlPULP is clocked from the PS, out of reset, and in idle state. Internal divisions of the external clock are handled within ControlPULP.

4. The PS drives ControlPULP booting process by flashing the L2 SRAM with the control firmware executable (Fig. 3.3).

5. The PS and PL start to asynchronously execute the plant simulation and the control firmware routines, respectively, through the shared memory.

The data are then collected for online dashboard monitoring, as detailed in the previous section.

The key benefit of the proposed methodology is the flexibility gained at the HW/SW interface. Design space exploration can be carried out on both the hardware controller and the control algorithm, with a short turnaround

[6] `https://github.com/EEESlab/examon`
[7] `https://buildroot.org/`

development time. The methodology especially fits integrated (on-chip) control systems validation and co-design due to the native on-chip hardware flexibility offered by the SoC-FPGA ecosystem.

3.5 Evaluation

In this section, we analyze and characterize both hardware (the ControlPULP platform) and software (the PCF) layers:

- We break down ControlPULP's post-synthesis area, which represents a small overhead ($< 1\%$) compared to a modern HPC processor die (Sec. 3.5.1).
- We first evaluate ControlPULP architecture with a cycle-accurate RTL testbench environment as depicted in Fig. 3.5. We model the latency of the interconnect network sketched in Fig. 3.1 by adding a programmable latency to the AXI4 interface. In the described test scenario, we first study the parallelization of the *control action* (P4)-(P7) on the cluster (Sec. 3.5.2). We then characterize *in-band* transfers, namely strided DMA accesses for data acquisition from PVT registers and CLIC interrupt latency with SCMI command processing (Sec. 3.5.3). Finally, we show the overall performance improvement of a single control step when accelerating control tasks in the cluster compared to single-core (Sec. 3.5.4). The testbench depicted in Fig. 3.6a does not provide the RTL description of the surrounding HPC processor. Instead, we model the closed-loop with a shared memory region between the PCS platform — the ISUT — and the system under control. Note that the real-time temperature and telemetry information from the HPC processor are pre-computed from a MIL emulation of the control algorithm executed with a fixed time step and statically stored in the simulation memory as unfolded in time.
- Since the standalone RTL simulation environment fails to provide a near-real-time closed-loop emulation framework for the control algorithm, we rely on the FPGA-SoC methodology detailed in Sec. 3.4 to analyze the PCF QoS when ControlPULP is mapped on real hardware and compare against a pure MIL approach to assess the validity of the HW/SW co-design.
- Finally, we show that the proposed PCF compares favorably against one of the most well-documented and freely accessible SoA industrial solutions on the market, IBM's OpenPower (Sec. 3.5.5). The comparison is carried in pure MIL simulation, being IBM's PCS hardware source publicly unavailable.

3.5.1 Area evaluation

We synthesize ControlPULP in GlobalFoundries 22FDX FD-SOI technology using Synopsys Design Compiler 2021.06. One gate equivalent (GE) for this

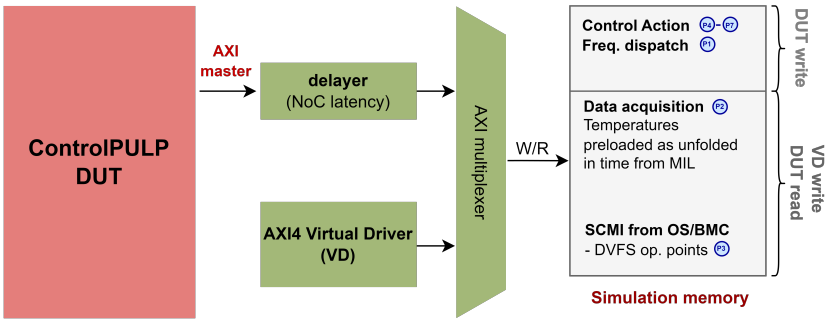


Figure 3.5: ControlPULP RTL testbench simulation environment

Table 3.3: ControlPULP post-synthesis area breakdown on GF22FDX technology.

Unit	Area [mm^2]	Area [kGE]	Percentage [%]
Cluster domain	0.467	2336.7	25.5
Manager domain	0.135	675.9	7.4
L1 SRAM	0.119	595.7	6.5
L2 SRAM	1.108	5542.1	60.6
Total	**1.830**	**9150.3**	**100**

technology equals 0.199 μm^2. The design has an overall area of 9.1 MGE, 32% occupied by the computing cluster accelerator and local L1 memory, and almost half (60%) by the main L2 memory in the *manager domain*.

We use a system clock frequency of 500 MHz. The rationale of the frequency choice is the following: the PCS typically belongs to the same clock domain of the CPU's *uncore* region. While the *core* region hosting the application-class PEs runs at frequencies in the order of 1-3GHz, the *uncore* is clocked at lower frequencies, in the order of hundreds of MHz, with modern architectures and technology nodes.

The target controlled-system die area is assumed comparable to other commercial, multi-core (> 64) server-class processors such as [64], about 460 mm^2. By correlating the gate-equivalent count of the HPC CPU die in the same technology node adopted for this work, ControlPULP would still represent about 0.5% of the available die area[8]. This first-order estimation makes the design choice of a parallel PCS valuable since its capabilities are much increased, while the silicon area cost remains negligible within a high-performance processor die.

[8]This has to be considered a first approximation, since it compares post-synthesis results with publicly available data of a modern HPC die, nowadays manufactured in a more advanced technology node.

3.5.2 Firmware *control action*

In the following, we analyze the execution of the PCF phases (P4)-(P7) on the multi-core cluster accelerator. We enforce power capping (*alpha* reduction [17]) to evaluate each computational phase fairly. Each cluster core is responsible for a subset of the controlled PEs. The parallelization is implemented as a fork-join process where the workload is statically distributed among the workers. In ControlPULP, the construct is implemented through a per-worker `thread_id` $\in [0 : N_{workers} - 1]$ and an equally distributed `chunk_size` where `chunk_size` $= \frac{N_{ctrl\_cores}}{N_{workers}}$. We are interested in extracting performance figures for the *control action* in a single periodic step n. We execute the PCF for S steps to amortize the effect of the initially cold instruction cache. Finally, we perform the arithmetic mean over S to get the mean absolute execution time for each (P4)-(P7) phase.

We report the execution time τ_0 and the multi-core speedup ($\frac{\tau_{0,single}}{\tau_{0,multi}}$) at varying number of controlled cores N_{cc} for each PCF phase in Figs. 3.6a and 3.6b respectively. The total speedup of the full *control action* at fixed N_{cc} is the geometric mean over the speedups of each phase. In our use case of 72 controlled PEs, ControlPULP executes the *control action* 5.5x faster than in single-core configuration, reaching 6.5x with 296 controlled cores.

We make the following observations. First, multi-core speedup scales with the number of controlled cores due to the increased workload and is affected by the workload characteristics of each phase. Second, the *control action* is not a fully computational step. In fact, instruction branching associated with power and frequency bounds checks per core introduces additional load/store stalls due to data access contention in a multi-core configuration. Finally, the computational body of (P6) and (P7) can be separated into independent parallel tasks and is thus an embarrassingly parallel problem. Instead, (P4) and (P5) show dependency across the values computed by the workers in the form of reduction sums, i.e., in (P4) to calculate the total power of the CPU and (P5) to calculate a normalization base for *alpha* power capping [17] and again the total CPU power. When a reduction sum is needed, we use a hardware barrier to synchronize the threads and join the concurrent execution on the cluster master core (core 0), which carries out the reduction.

As discussed in the analysis above, the increased parallel compute capability of handling the control's computational workload, paired with the general purpose nature of the accelerator, enable us to (i) improve the control performances with more advanced algorithms and (ii) be fully flexible when designing the control algorithm.

3.5.3 *In-band* Services

PVT sensors

To assess *in-band* services involving PVT physical sensors — phases (P1) and (P2) —, we measure the transfer time required for reading data bursts on

Table 3.4: Interrupt latency from interrupt edge to the first instruction in the interrupt handler as number of cycles.

Location	Increment [cycles]	Sum [cycles]
CLIC input to output	1	1
CLIC output to core (handshake)	2	3
Claim interrupt	1	4
Jump in vector table to CLIC handler	2	6
Save caller save regs (`addi` + 15 regs)	17	23
Compute and load CLIC handler address	5	28
Jump to CLIC handler address	2	30
Summary	-	**30**

the AXI4 master bus with the SoC timer. The exploration is three-fold: (i) direct data gathering from the ControlPULP cluster's cores, (ii) data gathering by offloading the transfers to the DMA in 1-D configuration, and (iii) DMA offload in 2-D configuration [104]. For (i) and (ii), we investigate the data collection on either 1-core or 8-cores configurations. The address range is equally distributed among the issuing cores in the latter scenario. In (iii), one core performs the read operation to highlight the advantages of offloading a single, large transfer with non-contiguous but uniformly spaced addresses to the DMA, which increases the addresses by the selected stride. This configuration becomes important when atomically gathering PVT information from equally spaced address locations (HPC PEs) with only one transfer request. As in Sec. 3.5.2, we use synchronization barriers to coordinate the eight cores. Fig. 3.6c reports the execution time τ_1 required for data movement when reading from up to 1000 PVT registers (4B each), an estimate bound given the number of PEs and the information needed from them (P, V, T, i.e., $\geq$ 3, lower bound). Fig. 3.6c shows that the best DMA-based transfers assuming 1000 PVT registers (2-D) are 5.3x faster than single-core direct data gathering.

SCMI and interrupt latency

An interrupt-driven (doorbell) transport regulates the communication of DVFS operating points in the agent-platform direction. Table 3.4 gives an overview of the overall CLIC interrupt latency measured as the number of cycles from the triggering edge in the CLIC to the ISR Handler's first instruction. The configuration of interrupt level, priority, and threshold configuration is handled with memory-mapped and CSR accesses to the CLIC register file and the manager core, respectively. This leads to a lower programming latency than software-driven approaches required in standard RISC-V platform-level interrupt controller (PLIC) or CLINT interrupt controllers.

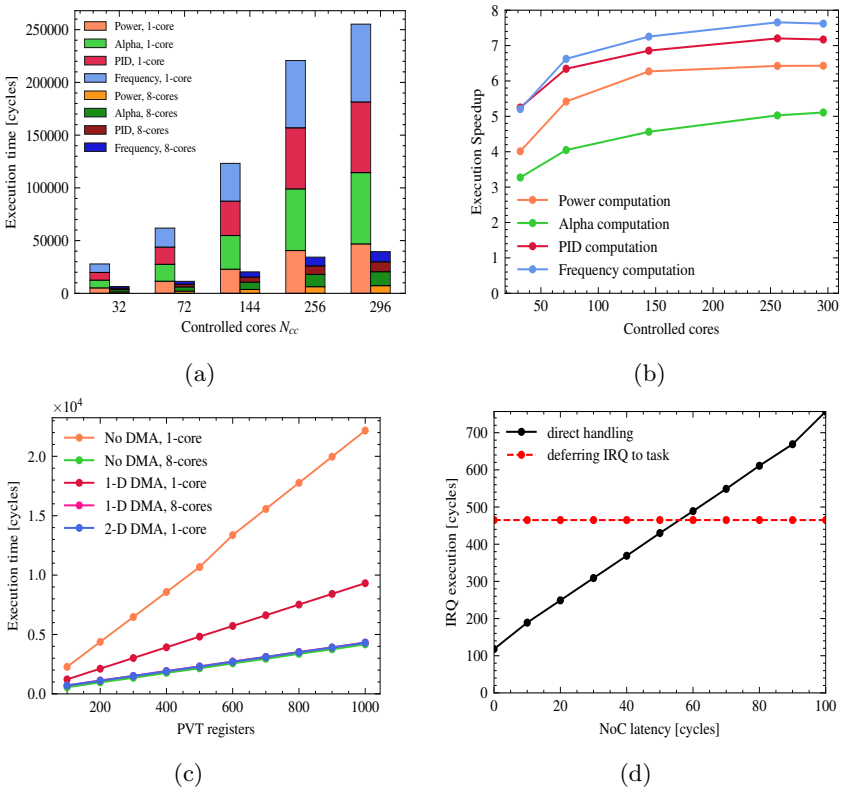


Figure 3.6: **(a)-(b)** Firmware *control action*, execution time, and speedup comparison between single-core (*manager domain*) and 8-cores (*cluster domain*). **(c)** *in-band* data acquisition from simulated PVT registers, execution time without and with DMA in 1-D and 2-D configurations. **(d)** Execution time in the interrupt handler from the interrupt edge to its completion with a basic SCMI message at varying interconnect network access latency to the mailbox

With a working frequency of 500MHz, the interrupt latency of one SCMI command coming from the OS HLC controller governor is negligible compared to the period of the PFCT that executes every 500μs, namely 0.01%.

Analogously, the context switch time needed to preempt the PFCT with the PVCT, which runs 4 times faster (every 125μs), during the execution of a PCF step is 0.08% the available time period of the PFCT, more than enough for the two tasks to coexists while executing their respective policies.

It is essential to notice that the latency of the interconnect NoC between ControlPULP and the mailboxes located in the die (Figs. 3.1) has a significant impact on the load/store access times, thus the time spent in the ISR, which grows with the interconnect delay size. We show this effect by emulating the shared mailboxes as well as the NoC latency in the RTL testbench environment (Fig. 3.5).

The black line in Fig. 3.6d reports the execution time for directly decoding and responding to a sample SCMI command (*Base Protocol*, `protocol_id = 0x10`, `message_id = 0x0` [12]) in the ISR when an external simulated driver rings a doorbell to the PCS. The figure reveals that the time spent in the ISR linearly increases with the NoC latency. We tackle the impact of NoC latency by deferring pending interrupts as they are triggered, thus keeping the ISR time short and insensitive to the CPU interconnect network delay, with the FreeRTOS timer API `xTimerPendFunctionCallFromISR()`. From the red line in Fig. 3.6d, we see that deferring interrupt handling to a task is preferable over direct handling, as it is network-latency insensitive for realistic NoC latencies larger than 50 cycles. Other existing solutions, such as Arm SCP firmware, propose a bespoke *Deferred Response Architecture* [1] to mark selected requests coming from an agent as pending and defer the platform response. We instead rely on a trusted scheduler that decouples OS and PCF driver APIs, improving flexibility and portability.

3.5.4 System-level PCF step evaluation

We finalize the standalone evaluations of ControlPULP's architectural features from the previous sections with the overall PFCT step cycle count comparison between accelerator-enhanced and single-core configurations, reported in Table 3.5 in the case of 72 controlled cores. Table 3.5 shows a breakdown of the required actions.

The total execution time differs in the two execution models. In the single-core case, we execute sequentially with less overhead from data movement ($T_{single} = \tau_0 + \tau_1$). In the multi-core case, ($T_{multi} = max(\tau_0, \tau_1) + \sum_{i=2}^{5} \tau_i$) we (i) execute the computation τ_0 and data acquisition τ_1 at step n concurrently, (ii) rely on $\tau_{0,multi} \ll \tau_{0,single}$, and (iii) introduce an overhead due to additional data movement involving L1 and L2 for data telemetry between *manager* and *cluster domains* during the PFCT.

Overall, multi-core execution achieves a 4.9x speedup over the single-core configuration. Provided a fixed *hyper-period*, i.e., the least common multiple of the control tasks' periods [98], which in the proposed implementation equals the PFCT step period of $500\mu s$, Fig. 3.7 shows the benefits of a programmable accelerator on the control policy time scale. Assuming a working frequency of 500MHz, and considering the interrupt latency and task preemption context switch time from the scheduler negligible as from Sec. 3.5.3, single-core execution time already leaves a free time window (*slack*) of about 70% the hyper-period with the reactive control algorithm implemented in this work. Cluster-based acceleration significantly raises the free time window to about 95% the PFCT period. This means the acceleration reduces the utilization time to complete the control task within the deadline. Furthermore, the more embarrassingly parallel the control problem, the more the concurrent speedup, hence the benefits of the acceleration.

Other effects in the plant can impact the control loop period in the control scenario presented in this chapter: (i) PLL lock time and voltage regulator

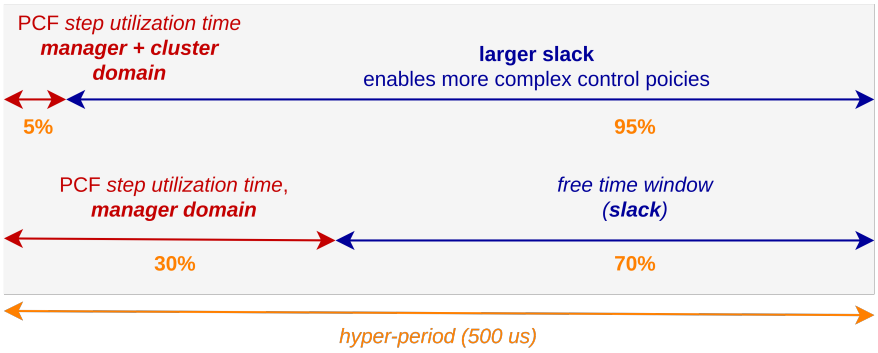


Figure 3.7: Benefits of a performant parallel PCS on the control problem, namely, room for more computationally-intensive control policies from the concurrent acceleration. The figure shows that the hyper-period free time window (*slack*) increases to almost 95% when the policy is accelerated with the *cluster domain*

response time, (ii) communication latency to sensors/actuators, and (iii) model dynamics evolution. PLL lock time and voltage regulator response time are in the order of microseconds. Similarly, communication latency depends on the interconnect network of the CPU. The PCF computes the frequencies/voltages at time step n and applies them at time step $n+1$. Therefore, latency issues - if any - due to the effects of (i) and (ii) can be addressed by starting early the application of the set points during time step n, provided the control period has enough slack time after the computation phase. Model dynamics (iii) are tightly coupled with the system in use and can be broken down into workload, power, and temperature effects. Temperature variations are in the order of magnitude of ms. Workload and power can potentially change at each clock cycle. Arm proposes the Maximum Power Mitigation Mechanism (MPMM) for mitigating and limiting the execution of high-activity events, such as vector instructions. The management of this mechanism is left to the power controller. This could be a potential extension to ControlPULP as well.

The $500\mu s$ hyperperiod used in the proposed implementation is shorter than the fastest dynamics in terms of temperature, but longer than the workload and power variations. The latter case represents a worst-case scenario to guarantee safe margins on the power constraint in actual HPC benchmarks.

3.5.5 Control-level PCF evaluation

We refer to the PCF QoS as an indicator of the control policy functional correctness (HW and SW) when the simulated MPSoC is assigned a certain workload. In this section, we present the results of this analysis by leveraging the FPGA-SoC closed-loop framework, the thermal and power model, and the synthetic workload introduced in sec. 3.4.3. We subsequently complete the QoS exploration with a comparison against the IBM OpenPower open-source control

Table 3.5: Execution time T of a PFCT step, single-core and cluster configurations. SCMI commands exchange and off-die transfers, handled by the SoC *manager core*, are not included in the comparison since they are a shared overhead.

Firmware phase	**Time step**	**Execution time [cycles]**		**Speedup**
		1-core	**Multi-core**	
control action (P4)-(P7)	τ_0	61867	11372	5.5x
in-band transfers (P1),(P2),(P3)	τ_1	5463	3523 (DMA)	1.6x
Offload to the Cluster	τ_2	-	389	-
L2 - L1 transfers	τ_3	-	434 (DMA)	-
L1 - L2 transfers	τ_4	-	872 (DMA)	-
Return from Cluster	τ_5	-	574	-
Step total time	T	67330	13641	**4.9x**

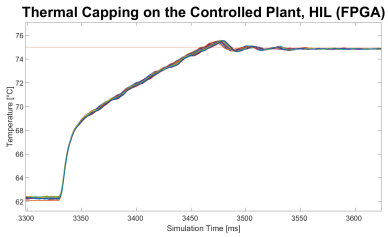
algorithm.

3.5.6 Standalone QoS evaluation on the HIL FPGA-SoC framework

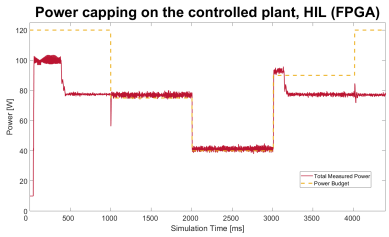
Provided a workload assigned to the simulated MPSoC, the HIL framework enables assessing the evolution in time of critical features of the control algorithm at the granularity of each controlled core (Sec. 3.3.1): power/thermal capping, workload-aware control, and frequency/voltage set-point tracking as a response to OS and BMC DVFS commands.

Figs. 3.8c and 3.8d show the frequency and voltage scaling enforced when the controller's SCMI mailboxes are notified DVFS operating point commands from the plant's *governor thread* running on the PS on a per-PE basis (Sec. 3.4.2). The SCMI agent dispatches the commands according to the executed workload; interrupts processing is deferred by the FreeRTOS scheduler and committed by ControlPULP's interrupt controller. Once registered, the governor's directive is processed by the PFCT in phases (P3)-(P7) and returns the reduced frequency as computed in the *cluster domain*. Fig. 3.8a and 3.8b show per-core thermal capping and the total power consumption and capping in action on the controlled plant during the execution of `Wsynth`, respectively. The maximum thermal limit from Fig. 3.8a, specific for each PE, is assumed to be 85 ℃. The orange line represents instead an additional thermal threshold required to stabilize the temperature with a safety margin in case of overshoots during the PID. Analogously, in Fig. 3.8b, the total power consumption of the system is bound by the power budget imposed through SCMI from the HLCs.

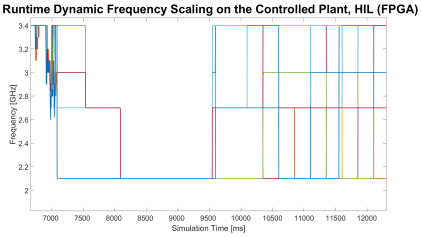
The HIL simulation is further compared with the software-equivalent model-based closed-loop from MATLAB Simulink, the first phase of the control algorithm design. Due to MATLAB Simulink runtime execution, we restrict the



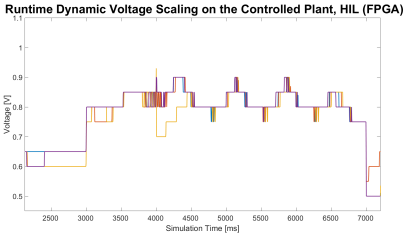
(a) Thermal capping on a subset of PEs.



(b) Total power consumption and capping action.



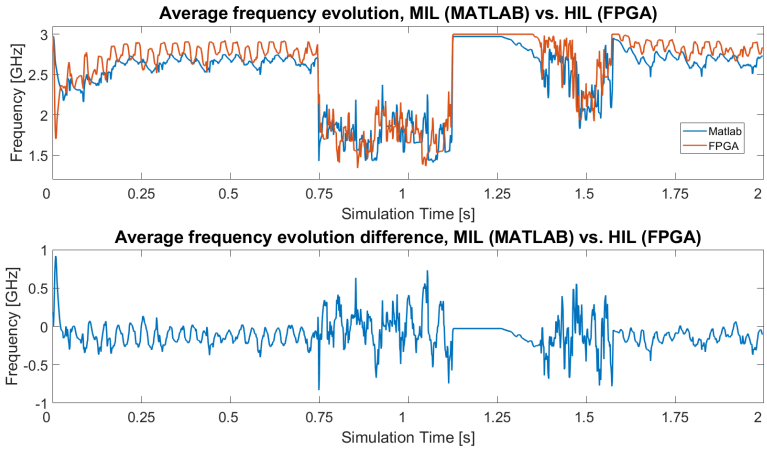
(c) Dynamic frequency scaling on a subset of PEs.



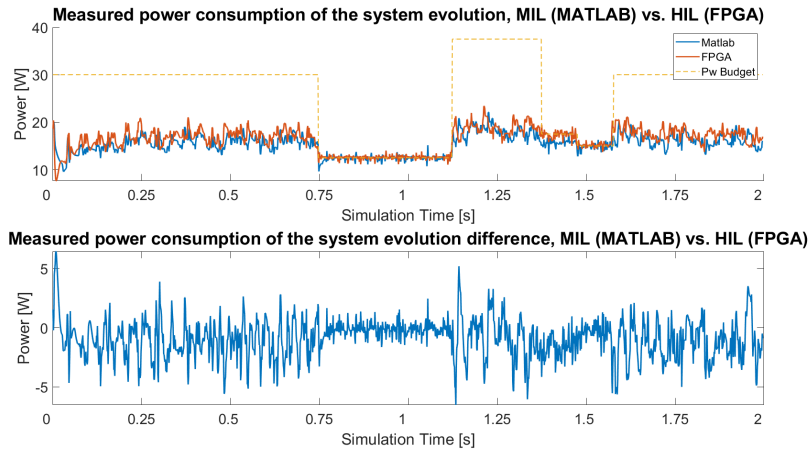
(d) Dynamic voltage scaling on a subset of PEs.

Figure 3.8: Thermal, power capping and DVFS emulation on the FPGA-SoC HIL framework with `Wsynth`. Requests coming from the HLCs (OS or BMC) governors, such as the target frequency and power budget, are processed by the reactive control, phases (P6)-(P7) described in Sec. 3.3.1. The number of controlled cores is 72.

floorplan of the controlled 72-core CPU to a tile of 9 cores, let the PVCT apply a constant voltage $V_{fixed} = 0.75V$ and fix the simulation duration to $2s$. Fig. 3.9a shows the average frequency evolution in time for the controlled cores in the tile. The discrepancy between the outcome of the two simulations has multiple reasons: (i) different data collection methodology: MATLAB records data at the exact time, while the FPGA simulation captures the information at a non-deterministic sequence of instants in the simulation interval due to its real-time characteristic, (ii) different resolution, and (iii) uncertainties and non-deterministic control delays introduced in the FPGA emulation and challenging to replicate in the MIL framework. Fig. 3.9b shows the measured consumed power of the assigned workload under power budget constraints from the HLCs, represented with a dashed yellow line. Overall, the HIL-based emulation gives comparable results when validated against the software-equivalent MIL. Albeit power spikes due to the discrepancies described above, DVFS tracking achieves a mean deviation within 3% the system's TDP (120W in the emulation), more than acceptable for the assessment.



(a) Evolution in time of the average frequency in the simulated tile.



(b) Evolution in time of the average consumed power in the simulated tile. The dashed yellow line represents power budget directives dictated by the HLCs.

Figure 3.9: HIL and MIL comparison when `Wsynth` is assigned to the controlled system. The emulation assumes a floorplan with a tile of 9 cores and runs for $2s$ to align with MATLAB runtime.

Comparative QoS evaluation with SOTA

We cross-benchmark the PCF with the IBM OpenPOWER (Sec. 3.2). We model IBM's control action, i.e., the *Voting Box Control* described in Sec. 3.2, excluding a few architecture-specific features, and the two-layer PCF control described in the proposed framework with MATLAB Simulink to enable a fair and hardware-agnostic comparison. The PID-like coefficients of the IBM control are adapted to the HPC chip model power and thermal characteristics. `Wsynth` (Sec. 3.4.2) is distributed as follows: *core 1*-*core 3* and *core 2*-*core 4* pairs are assigned `WsynthMax` and `WsynthIdle` respectively. *Core 5*, *core 6*, and *core 9* execute `WsynthMix` while *core 7* and *core 8* are exposed to `WsynthFast`. We rely again on constant voltage $V_{fixed} = 0.75V$ and do not consider overhead nor delays in the PLLs and VRMs operating point transitions. The power budget is changed five times during the simulation to stress all the elements of the control action. The simulation runs for $2s$ on a tile of 9 cores.

First, we show that a controller with a multi-core cluster able to deliver higher computational power is beneficial to the performances of the HPC chip. We compare the IBM control and a version of it with a per-core temperature PID for frequency reduction. In fact, as from Sec. 3.2, the IBM control policy considers the maximum temperature among the PEs when applying frequency reduction. We conjecture that this limitation is enforced by the limited control policy complexity that can be handled by IBM's OCC. Conversely, ControlPULP enables fine-grained frequency reduction on a per-core temperature granularity. The performance is shown in Fig. 3.10. The number of retired instructions indicates the execution time achieved by the workload at the end of the simulation: the more retired instructions, the faster the workload, meaning a more efficient control.

While using only one temperature for the whole tile results in an average performance reduction per core of 5%, cores executing high-power instructions (*core 1* and *core 3*) receive a performance increase of 4% and 5% respectively. In fact, being the frequency reduction based on the hotter cores and thus a shared penalty, neighboring cores get colder, and other cores consume less power during power capping phases, leaving more power available to boost performances of *core 1* and *core 3*.

Last, we compare the PCF and the IBM control with per-core temperature PIDs. The PCF control favors cores executing high-power instructions [17] (*core 1* and *core 3* in this simulation with `Wsynth` benchmark), thus compensating the performance penalty showed in the previous test. Results in Fig. 3.11 show a performance increase in executed instructions ranging from +2.5% to +5%. This holds for cores with mixed instructions (up to +3.5%) as well, while cores involved in less demanding workloads witness a decrease between −2% and −3%. We conclude that the modified policy with per-core temperature PID calculation can selectively boost the retired instructions, achieving a higher application performance on the HPC chip while still meeting the thermal cap.

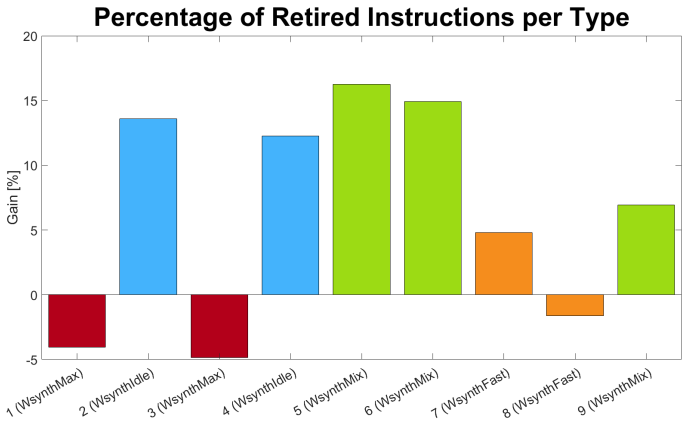


Figure 3.10: Comparison between the modified IBM OpenPOWER control with per-core temperature PID for frequency reduction and the original IBM OpenPOWER control. The simulation time is $2s$

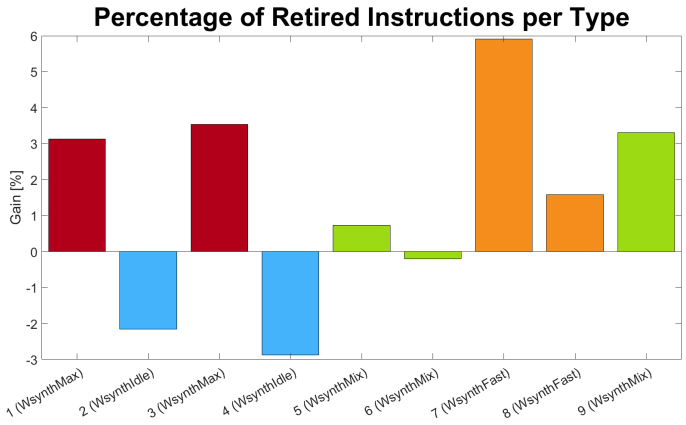


Figure 3.11: Comparison between the PCF control and the modified IBM OpenPOWER firmware with per-core temperature PID for frequency reduction. The simulation time is $2s$

3.6 Conclusion

This chapter presented ControlPULP, the first HW/SW RISC-V control system for HPC processors that exploits multi-core capabilities to accelerate control algorithms, featuring a specialized DMA and fast interrupt handling and synchronization. The HW/SW was evaluated while accounting for physical state variations surrounding the integrated controller by designing an agile, near-real-time closed-loop emulation framework that takes inspiration from the road paved by modern HeSoC platforms on FPGA. The framework relies on a power and thermal model of the controlled CPU as plant, which is paired with workload instruction traces and the control algorithm to realize the closed-loop. With the proposed multi-core architecture, a control policy step executes 4.9x faster than in a single-core configuration, increasing the hyper-period's *slack* and thus enabling the implementation of more complex and computationally demanding control algorithms (for example, predictive policies) with fine-grained frequency dispatching.

Chapter 4

Performance Characterization of Power-Management Interfaces in HPC Processors

4.1 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 [15]. 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 [15]. 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 [56]. 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 OS running on top of AP. The agent handling PM in the OS is known as OSPM [49]. 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 SoC, coupled with additional safety and security requirements, might be too complex for the OS to manage while simultaneously optimizing workload performance [73]. 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 CPU thermal constant, power and current can vary so quickly that OSPM SW is unable to deal with them timely [49].

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 LLCs. The implications of such a model were not fully taken into account when industry-standard PM FW, such as ACPI described in section 4.2.4, was first proposed. Throughout the years, HPC and mobile platforms from leading industry competitors started adopting this model with proprietary HW/SW architectures [88].

A delegation-based scheme implies clear responsibility boundaries between the HLC and the LLC. Therefore, a key design element becomes the 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 DVFS policies according to application performance patterns and the underlying architectural power profiles [109, 36]. Additionally, some efforts have concentrated on control algorithms to dynamically adjust voltage and frequency operating points within multicore systems [62]. 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.

The study presented in this chapter focuses on Arm-based server systems and employs the open-standard SCMI protocol as a representative example. 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 4.2.4.

The assessment was carried out using an open-source HW/SW LLC design, coupled with a FPGA implementation of a HIL framework for power and thermal management simulation [88]. 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 4.3.

To the best of the author's knowledge, the study presented in this chapter provides the first fine-grained and quantitative analysis of modern PMI behavior and its impact on runtime power management. This chapter is based on the journal publication [119], whose main contributions are summarized below:

1. **Integration of an FPGA-based hardware-in-the-loop framework.** The FPGA-based HIL platform introduced in chapter 3 [88] was extended to analyze the communication backbone between high-level OSPMs and the hardware and firmware of the LLC. This was achieved by integrating the complete Linux SCMI software stack and implementing a dedicated SCMI mailbox unit (SCMI-MU) compliant with the message handling unit (MHU)-v1 specification.

2. **Quantitative characterization of communication latency.** The performance of hardware/software low-level PMIs in Arm architectures was evaluated in terms of latency metrics. The dispatch time of SCMI messages through the Linux OSPM stack was measured to be 70.5 μs, while the average processing time for SCMI response messages was 603 μs.

3. **Analysis of control-loop performance and firmware optimization.** Using the developed setup, the entire power-management control scheme was assessed under different configurations—namely, periodic and event-driven high-level control. The study quantified how latency within the PMI affects end-to-end **control-loop performance**, and demonstrated that optimized firmware can reduce the overall round-trip latency from approximately 1.3 ms to 114 μs. This improvement results in up to 3% faster application execution time for the evaluated representative workload.

4.2 Background and Related Works

4.2.1 Overview and Terminology

In this section, we recall the main terminology and HW/SW components for PM of modern HPC SoCs, pictured in fig. 4.1. The taxonomy is based on the three largest domains—HLC, LLC, and their interface.

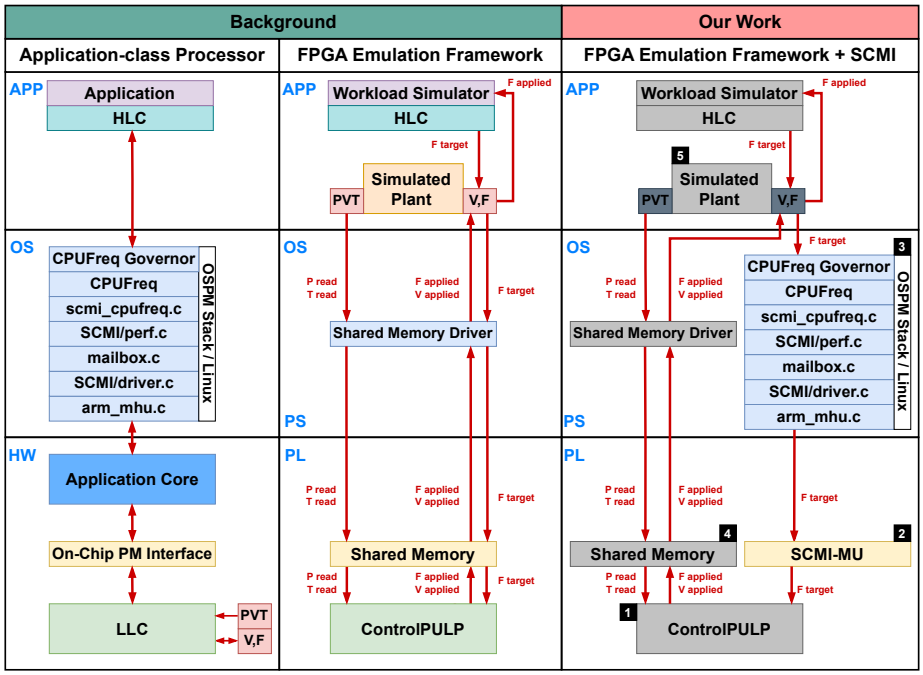


Figure 4.1: Overview of the main PM components of modern HPC SoCs. The figure highlights the PMIs linking HLCs to the LLCs from a HW and SW perspective in Arm-based server-class CPUs. (**a**) Architecture of the power management scheme in an application-class processor, (**b**) Block diagram of a state-of-the-art HIL platform for power management systems emulation, (**c**) Block diagram of the extended HIL platform proposed in this chapter.

The UEFI's PM ACPI specification distinguishes among three types of PM: *system*, *device*, and *processor* [117]. Similarly, Arm differentiates between *system*, *device*, and *core* PM services [6]. *Processor* (or *core*) and *device* services are typically requested by the OSPM, while *system* services are provided without OSPM mediation, for example, by the BMC on the motherboard [6]. This chapter 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 PE.

The LLC periodically interfaces with 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 4.2.4. These on-die interactions are collectively known as *in-band* services [88, 21], 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* [21]. 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.

4.2.2 HLC Components

Hardware Layer

Modern server-class SoCs are heterogeneous many-core chiplet architectures. Each chiplet integrates tens to hundreds of AP PEs, 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.

Power Management Software Stack

The HLC's OSPM is the agent that controls the system's power policy [49]. We rely on terminology adopted in the Linux OSPM, used in this chapter. 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 in practice, selecting the highest available 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.

4.2.3 LLC Components

Hardware 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 PMCAs 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 [16]. Moreover, its I/O interface has to sustain *out-of-band* communication through standard HW PMIs, such as PMBUS [113] and AVSBUS [112].

Power Management Software Stack

The PM policy can be scheduled as a bare-metal FW layer [102, 7], or leverage a lightweight RTOS (e.g., FreeRTOS [19]), as in the open-source ControlPULP [88] employed in the work presented in this chapter. 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 [122], as reported in fig. 4.1b.

4.2.4 HLC-LLC Interface

Industry-Standard Firmware Layer

ACPI [117] 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.

OS-Agnostic Firmware Layer

Most industry vendors rely on proprietary interfaces between the HLC and LLC. Intel leverages model-specific registers (MSRs) mapped as ACPI FFH to tune PEs' performance, where performance requests are handled through the `intel_pstate` governor in Linux, while power capping is enforced via 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 within the OCC to exchange control and telemetry data with the LLC.

Arm, on the other hand, avoids the limitations imposed by FFHs and introduces the System Control and Management Interface (SCMI) protocol to handle performance, monitoring, and low-power regulation requests between the HLC and LLC. SCMI defines 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 commands through a shared memory area (mailbox unit) and issues responses according to a standardized message-based protocol. The design of SCMI reflects the industry trend of delegating power and performance management to a dedicated subsystem [73], providing a flexible and platform-agnostic abstraction layer.

A specification with analogous goals is currently being defined within the RISC-V ecosystem: RPMI [101]. RPMI shares the same architectural principles and objectives as SCMI, offering a standardized communication channel between the operating system and the platform management controller for power, performance, and telemetry control. Both interfaces rely on message-based

communication over shared memory regions, synchronized through doorbell-like signaling mechanisms to trigger command and response exchanges. While SCMI is a mature and widely adopted standard within the Arm ecosystem, RPMI is an open specification still under active development. Given the strong conceptual and structural similarity between the two protocols, the latency characterization presented in this work for SCMI is directly applicable to future RPMI-based implementations.

Despite the widespread adoption of proprietary HLC–LLC interfaces in current commercial x86-based platforms, a quantitative and systematic comparison of the communication latency across different power-management interfaces is currently not possible. While register-based mechanisms such as MSRs and SPRs enable low-latency access to the firmware interface, vendors typically expose and document only end-to-end power and performance management reaction times, which include internal firmware execution and platform-specific control loops. The latency contribution of the HLC–LLC communication interface itself is therefore not publicly characterized. As a consequence, existing studies do not allow isolating and evaluating the communication overhead of HLC–LLC interfaces. This work addresses this gap by providing an experimental characterization of the latency introduced by a standardized HLC–LLC communication interface for power and performance management. Specifically, we analyze the performance cost introduced by delegation-based architecture in modern PM frameworks. We first characterize the intrinsic latency incurred when traversing the HW/SW hierarchy —from the HLC to the LLC—and quantify its impact on end-to-end control responsiveness. Building on this characterization, we then propose a firmware-level optimization strategy that exploits the measured SCMI latency to improve overall PM efficiency. The following section introduces the experimental methodology used to perform this co-design.

4.3 Methodology: HIL Framework on FPGA

fig. 4.1c 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 RTL HW description (❶). ControlPULP is emulated in the PL of the FPGA-SoC, to which we added a SCMI-MU (❷) to provide the HW transport for SCMI protocol. At the same time, the Linux OS image (❸) running on the PS has been modified to propagate OSPM requests to the ControlPULP LLC via the SCMI-MU. Additionally, a shared memory interface (❹) is in place to emulate PM virtual sensors and actuators. Indeed, the simulated plant (❺) 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.

4.3.1 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 [5], ensuring alignment with the corresponding low-level drivers in the Linux system. As shown in fig. 4.1, the SCMI-MU is accessible from both the PS and the LLC through a 32-bit AXI Lite front-end [1]. 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 [73], 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 CLIC, which handles ControlPULP hardware interrupts and improves real-time performances [16]. On the HLC side, the completion interrupts are routed through the Arm GIC-V2 [10]. 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.

4.3.2 Linux SCMI Software Stack

As mentioned in section 4.2.2, 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 fig. 4.2 shows the interaction between the Linux kernel drivers upon the arrival of a new target frequency from the user. 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 4.2 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`.

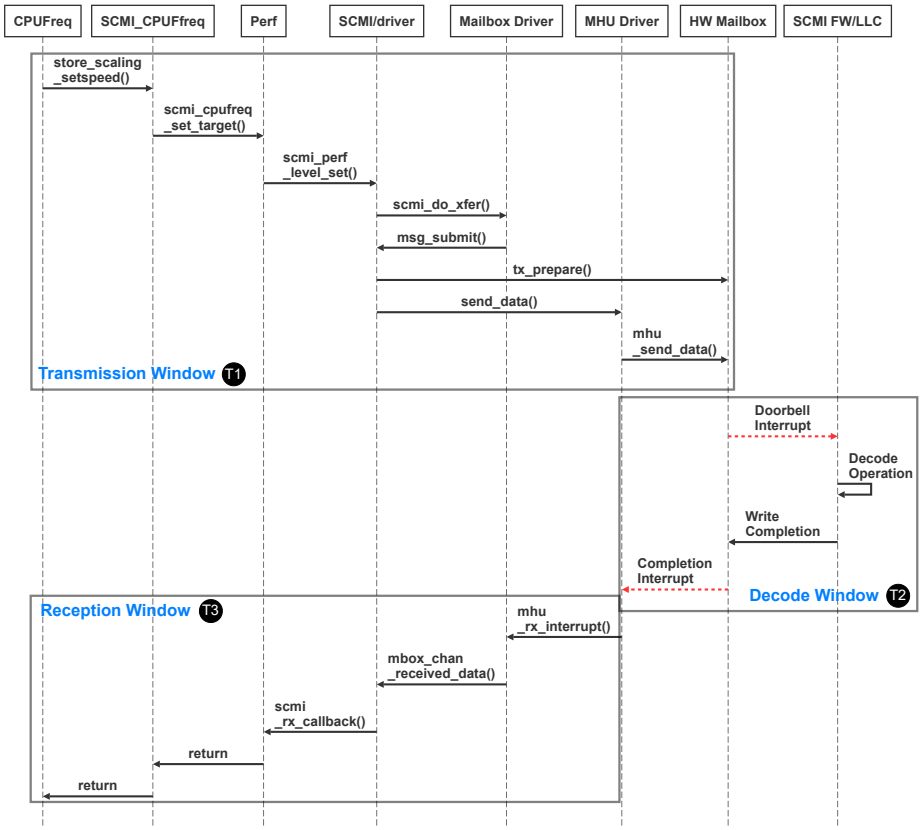


Figure 4.2: Sequence of function calls within the Linux CPUFreq stack, for a perf_level_set SCMI request.

4.3.3 LLC SCMI Firmware 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 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.

4.3.4 LLC Power Management Policy Optimized for Latency

In this chapter, we propose an LLC PM policy optimized for latency. The default ControlPULP PM policy (PCF) [88] relies on a periodic control task executed every 500 µs (configured to account for ms-scale temperature time constant and µs-scale 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 µs period to compute the novel power management settings (PLL frequencies and voltage level), which are 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.

4.3.5 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 fig. 4.1. 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}, $f^i_{T,max}$, 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 [90], $f^i_T[k]$ read as follows [20]:

$$f^i_T[k] = \frac{f^i_{T,max} \cdot N_{ic}[k-1] \cdot CPI_{ic}[k-1]}{N_{im}[k-1] \cdot CPI_{im}[k-1] \cdot (\beta - 1) + N_{ic}[k-1] \cdot CPI_{ic}[k-1] \cdot \beta} \tag{4.1}$$

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.

4.4 Evaluation

section 4.4.1 characterizes the SCMI call stack and evaluates its latency. section 4.4.2 evaluates the impact of PMIs on an end-to-end PM policy running on FPGA-HIL.

4.4.1 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 4.3.1, namely, from calling `store_scaling_setspeed()` to `cpufreq_notify_transition()`. The second focuses on the LLC FW, measuring its latency from the SCMI notification.

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 µs. We applied it to 1000 frequency setpoint requests. In all the tests conducted in this in this evaluation, the PS was configured to run at 1.2 GHz.

The total time for executing a `perf_level set` request is detailed in table 4.1. We show minimum, maximum, and average time across the 1000 setpoint requests sent to the LLC. From table 4.1, 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 fig. 4.3. We observe that T1 exhibits a probability distribution falling within a narrow range of only 3 µs, so we consider its mean value of 70.5 µs as relevant. T2 shows an almost uniform distribution in the range between 83 µs and 1205 µs, while for T3, we can notice a bimodal distribution with two peaks centered around 75 µs and 164 µs.

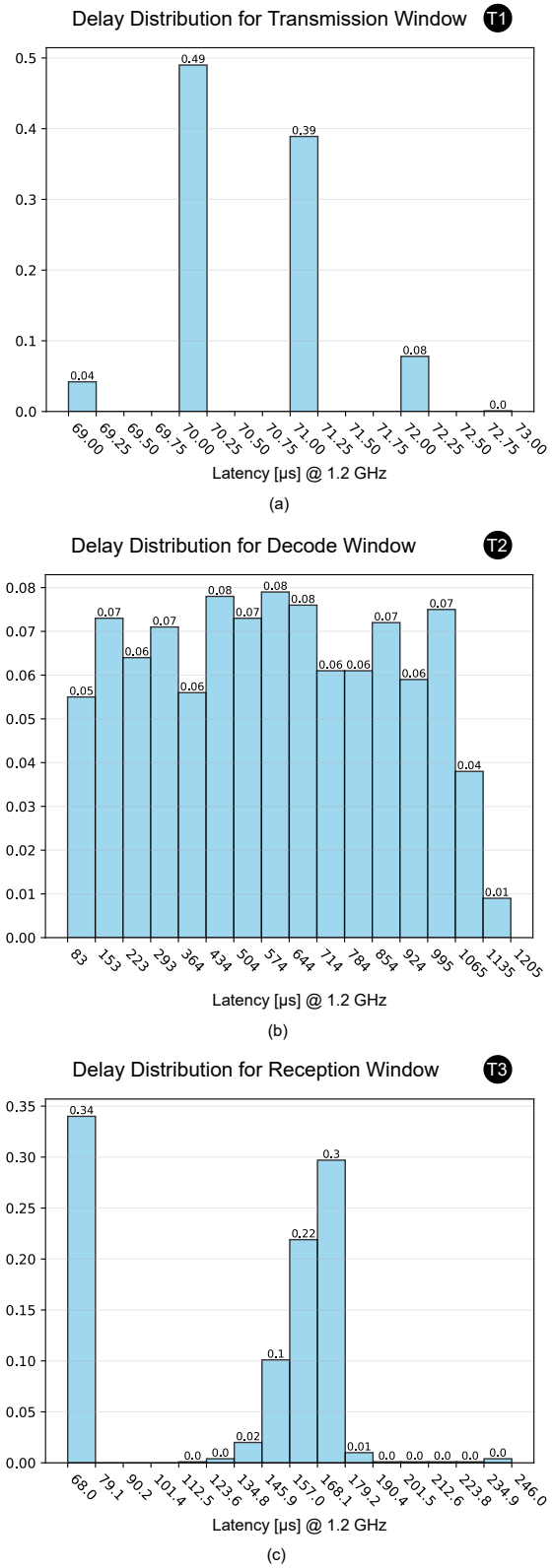


Figure 4.3: Delay time distribution for (**a**) transmission window, (**b**) decode window, and (**c**) reception window.

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. To further interpret the observed variability in T2, we also analyzed the interrupt routing behavior of the SCMI transport on the target platform. During the latency characterization runs on the ZCU102 board [122], the interrupt associated with the SCMI mailbox doorbell was always handled by the same CPU core, as observed from the Linux interrupt accounting table [97]. Therefore, the measured jitter in T2 cannot be attributed to interrupt migration across cores, and is instead mainly explained by the OS wake-up and scheduling latency of the software components involved in the SCMI transaction handling, as well as by contention on shared processing resources on the core serving the interrupt. These effects reflect realistic operating conditions of Linux-based power management stacks, where power-management activities execute concurrently with other system workloads.

Table 4.1: SCMI call stack time measured at 1.2 GHz.

	Minimum		**Average**		**Maximum**	
	[µs]	**[cycles]**	**[µs]**	**[cycles]**	**[µs]**	**[cycles]**
T1	69	82.60k	70.50	84.60k	73	87.60k
T2	83	99.60k	603.50	724.20k	1205	1446k
T3	68	81.60k	133.80	160.56k	246	295.20k
Total	220	264k	807.80	969.36k	1449	1828k

Table 4.2: ControlPULP SCMI decoding time @20 MHz.

	Increment		**Sum**	
	[µs]	**[cycles]**	**[µs]**	**[cycles]**
CLIC to ISR	2.30	46	2.30	46
ISR exec.	5.20	104	7.50	150
ISR to dec. task	0.65	13	8.15	163
Dec task exec.	5.00	100	13.15	263

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 [106] 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 4.2 with respect to the 20 MHz frequency of ControlPULP in the HIL's framework. In the table, the first row reports ControlPULP's CLIC latency, [88]. The total LLC decoding time takes 263 clock cycles, i.e., 13.15 µs @20 MHz.

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 MHz, 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.

4.4.2 Characterization of End-to-End Power Management 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 $f^i_{T,max}$) and memory instructions, whose throughput depends solely on memory subsystem speed. This synthetic workload provides clearly defined and repeatable phase transitions, enabling an accurate and reproducible evaluation of the timing behavior of the end-to-end power management loop in response to workload changes.

We compare the proposed SCMI interface with an ideal near-zero latency shared memory transport that bypasses the OS. All time-related quantities referring to the workload execution and to the control behavior are expressed in the emulated time domain of the HIL framework. 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 $20ms$ each and lasts $10s$ when executing at the maximum frequency ($f^i_{T,max} = 3.4$ GHz). 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 GHz during memory phases and the maximum one during compute phases. The tests conducted with a periodic HLC are configured with $T_{f_T} = 1$ ms. With the average SCMI communication latency being 70.5 µs, which is almost an order of magnitude lower than the execution periodicity of both the control task and the communication task ($T_c = 500$ µs) [88], most of its effects on the control performance are masked in the LLC. For this reason, we restrict the evaluation to $T_{HLC} < T_{WL}$. We account for this difference in the emulation setup by introducing a real-time delay of 1.7 ms when forwarding new target frequencies to the SCMI channel, so as to reproduce, in the emulated time domain, the measured SCMI communication latency.

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 4.3.4. The event-driven HLC algorithm used in test (ii) is derived from previous work that optimizes DVFS policies based on workload patterns [109, 36]. The average CD for (i) is $2.20ms$ for SCMI communication and $1.94ms$ for the shared memory case. Test (ii) results in an average CD of $1.35ms$ with SCMI communication and $1.16ms$ 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 $0.77ms$. 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 $1.42ms$, whereas for frequency decreases, the CD is significantly lower, reaching just 114 µs. fig. 4.4 shows the distribution of the CD during (i) and (ii), (i)-fig. 4.4a,b, and (ii) fig. 4.4c,d.

To ensure consistent and reproducible experimental conditions across all the evaluated configurations, the application speedups reported in this section are obtained using the same synthetic workload adopted for the control-delay analysis. While absolute speedup values may vary for real HPC applications exhibiting irregular, communication-intensive, or memory-bound execution patterns, the observed trends across different HLC and LLC configurations capture the relative impact of the proposed control strategies and of the SCMI communication latency on the end-to-end power management behavior. Accordingly, table 4.3 reports the application execution-time speedup for the same three test configurations with respect to the periodic HLC and the baseline ideal shared-memory PMI. From the reported results, we observe that the event-driven HLC (ii) leads to a speedup of +2.75%, which further increases to +3.18% with the proposed LLC-optimized FW. Moreover, the CD introduced by the SCMI protocol leads to only a marginal reduction in the attained speedup (from −0.02% to −0.3%). These speedups are enabled by a more timely reduction of frequency during memory-bound phases, which improves the alignment between workload phases and operating points and allows the LLC FW to exploit the locally available power and thermal headroom of the controlled cores more effectively during subsequent compute-intensive phases, while still enforcing power and thermal

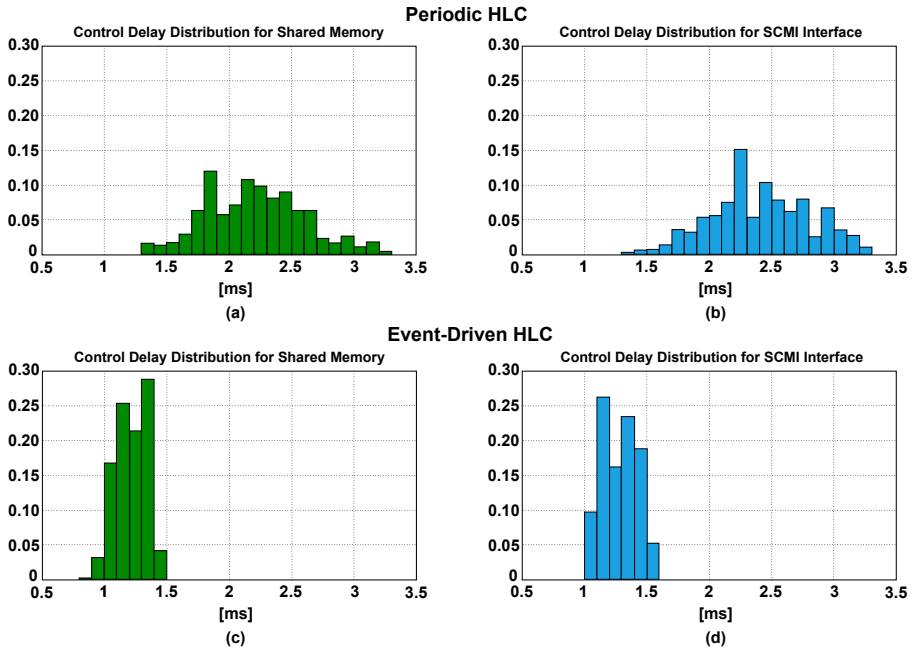


Figure 4.4: Control Delay distribution for different HLC configurations and interfaces: (**a**) periodic HLC with shared memory, (**b**) periodic HLC with SCMI mailbox, (**c**) event-driven HLC with shared memory, (**d**) event-driven HLC with SCMI mailbox.

caps.

Table 4.3: Execution time speedup over different HLC and control algorithm configurations.

	Execution Time Speedup [%]	
Configuration	**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

4.5 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 µs at 1.2 GHz 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 µs (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 µs and 1.2 ms at 1.2 GHz 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, resulting in an application time reduction of around 2.7% for the execution of the evaluated workload.

These results show that the end-to-end latency of the PM software stack, from the application layer down to the OS drivers and the PCU firmware, is a key factor in determining the responsiveness of the power-management loop and its impact on application execution. In particular, our analysis highlights that the relative contribution of the different components of the control stack—namely HLC policies and LLC control FW—dominates over the communication interface itself in shaping the observed control delay. 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 can enable higher energy savings. As an example, work from Cesarini et al. [34], 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 µs in Intel systems. Our proposed approach lowers the command propagation latency from 1.35 ms to 114 µs 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.

4.6 Conclusions

In the work presented in this chapter, we extended the FPGA-based HIL platform introduced in [88], 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 µs at 1.2 GHz, attributed to the dispatching of an SCMI message through the Linux OSPM stack and an average of 603 µs 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 µs, resulting in an application execution-time reduction of approximately 3% for the evaluated workload.

Chapter 5

Towards User-Space Power-Management Communication Interfaces

5.1 Introduction

Modern monolithic processor systems typically adopt a *delegation-based* paradigm for power management, where a high-level OSPM issues *performance requests*—i.e., target frequency levels for each performance domain—to a LLC responsible for enforcing the selected operating states. The LLC, usually implemented as a microcontroller with sufficient computational capability, executes control loop firmware that configures the processor *operating points*, expressed as frequency–voltage pairs for individual cores, uncore domains, and other functional units. The overarching objective is to minimize performance degradation while ensuring operation within the physical constraints imposed by the processor's physical design, such as limits on power, temperature, and current.

In contemporary HPC processors, a dedicated PCS typically undertakes these low-level power-management duties, aggregating telemetry from distributed on-chip sensors and applying the requested performance states. Within the delegation-based scheme, the OSPM and the controller exchange two primary classes of information: (i) performance requests flowing from the OSPM to the controller, and (ii) telemetry flowing back from the controller to the OSPM, including domain temperatures, voltage levels, and current performance states. These exchanges are mediated by dedicated power-management interfaces that couple the operating-system policy layer with the HW control layer.

In current platforms, such interfaces typically rely, at the HW level, on dedicated shared-memory regions or MSRs to enable bidirectional data exchange [11]. At the SW level, communication is commonly handled by kernel-resident power-management governors bound to kernel drivers, which both expose the com-

munication protocol to upper layers and implement low-level HW calls. These drivers usually embed the protocol specification (message formats, handshake rules, admissible telemetry), with prominent frameworks including ACPI [117], SCMI [11], and RPMI [100]. Although certain specifications allow adding new message classes (e.g., additional control commands or telemetry topics), such extensions still require kernel-space modifications, complicating integration and debugging and tying deployments to specific kernel versions and driver stacks.

As processor architectures evolve from single monolithic dies to *chiplet-based* designs, each chiplet often integrates a local controller and distinct power domains. The delegation-based model therefore shifts from a single centralized entity to a *distributed* arrangement with multiple controllers—one per chiplet. Beyond the standard OSPM-controller exchanges and the on-chip/off-chip interfaces to local resources (e.g., PVT sensors, BMC, VRMs), this paradigm requires robust synchronization among multiple controllers. Today, inter-controller synchronization is typically implemented over existing on-chip communication fabrics (e.g., AXI [9], APB [8], or proprietary control interconnects), which are highly design-dependent and do not scale gracefully as the number of chiplets, and thus controllers, grows.

To address these limitations, this chapter presents a novel user-space communication approach that moves the entire OSPM protocol stack to user space via a *shared-memory* transport compliant with the *Micro XRCE-DDS* [44] communication middleware.

The proposed interface is implemented on *ControlPULP*, a RISC-V–based PCS, and extends across both the host processor's OSPM and the controller. On the OSPM side, the communication stack is relocated to *user space*, enabling message exchange without kernel-resident protocol logic. On the controller side, a lightweight implementation of the same interface is deployed within the PCS, where the *Micro XRCE-DDS* Client executes the corresponding low-level communication tasks. This co-design establishes a user-space managed communication path between the host and the controller, enabling future integration of policy managers and telemetry services without kernel mediation, and remaining independent of kernel versioning and driver coupling on the host processor. Furthermore, the Micro XRCE-DDS middleware inherently supports distributed networks of endpoints (topics, publishers, and subscribers), providing a scalable substrate for coordination and synchronization among multiple controllers within the same processor.

The following contributions are presented in this work:

1. **Integration of Micro XRCE-DDS to the ControlPULP PCS (RISC-V).** The communication middleware is integrated into ControlPULP, a PCS based on the RISC-V Instruction Set Architecture (ISA), enabling standard-compliant publish/subscribe semantics on a resource-constrained controller.

2. **Shared-memory transport for user-space OSPM–controller communication.** A shared-memory communication interface transport layer is designed and implemented, compliant with Micro XRCE-DDS middleware.

3. **Execution-loop latency and memory-footprint evaluation.** A quantitative assessment of Micro XRCE-DDS Client firmware message dispatching loop cycles and stack usage on ControlPULP is provided as a function of the number of middleware topics, characterizing scalability and resource requirements.

5.2 Related Works

5.2.1 Kernel-mediated power-thermal interfaces

Power and thermal management on contemporary platforms is commonly mediated by kernel-resident frameworks that expose high-level controls to the operating system while encapsulating low-level HW access. *ACPI*-based designs provide declarative firmware tables and kernel drivers to negotiate performance, idle, and thermal states, with policy typically implemented in user space but enforced through kernel interfaces [117]. *Arm SCMI* generalizes this model with a message-based protocol exposing services for performance, power, clock, and sensors; Linux integrates a core SCMI stack and per-service drivers, with optional extensibility via vendor-specific channels [12]. Policy daemons such as *thermald* [59] and platform frameworks like *Intel DPTF* [58] execute userspace control loops while relying on kernel drivers for actuation and telemetry. Similarly, the *CPUFreq* [67] subsystem supports a userspace governor, yet the mechanism for frequency transitions remains a kernel responsibility.

These approaches standardize interfaces and safety invariants, but they tightly couple message formats, handshake semantics, and data models to kernel versions and driver stacks. Adding new telemetry streams or control transactions typically requires kernel-space development, complicating integration, testing, and portability across distributions and platform releases.

5.2.2 Inter-controller synchronization in multi-chiplet systems

As designs shift from monolithic dies to multi-chiplet processors, power management becomes a distributed problem in which per-chiplet controllers must coordinate budgets and states. Conventional implementations often employ platform-specific buses [9],[8] and ad hoc protocols, which are design-dependent and scale poorly as the number of controllers grows. Kernel drivers again mediate access, constraining the evolution of synchronization semantics and hindering reuse across silicon generations. Moreover, such solutions remain largely design-dependent, tightly coupled to packaging technology and platform firmware, and neither decouple the control plane from kernel mediation nor provide a scalable, reusable synchronization substrate as controller counts grow.

5.2.3 Micro XRCE-DDS

Micro XRCE-DDS is the Data Distribution Service (DDS) [85] profile for eXtremely Resource Constrained Environments, providing typed publish/subscribe semantics with a lightweight Client–Agent architecture [44]. In common deployments, microcontroller *Clients* expose publishers/subscribers and communicate with a host-side *Agent* that handles discovery, type registration, and bridging to full DDS/Real-Time Publish Subscribe (RTPS) domains. Transports are pluggable and typically include UDP/IP [96],[95], and serial links (e.g., UART), selected according to HW and SW constraints. This middleware enables MCU endpoints to participate in standardized data exchange (topics, types, QoS) with a modest footprint; however, host paths in practice traverse kernel-managed networking stacks, leaving room for designs that minimize kernel mediation on the control data path.

5.2.4 Summary

Kernel-mediated power-thermal interfaces (ACPI, SCMI, DPTF, CPUFreq) offer mature safety and interoperability but are rigid to extend: adding message types or telemetry typically entails kernel-space changes tightly coupled to specific versions. In heterogeneous and multi-chiplet systems, host–controller messaging and inter-controller synchronization remain funneled through kernel endpoints or platform-specific fabrics; even with emerging D2D control links, solutions tend to be design-dependent and do not by themselves standardize control semantics or remove kernel mediation on the critical path. Meanwhile, Micro XRCE-DDS provides a typed, lightweight pub/sub model for constrained controllers.

The approach considered in this chapter places protocol handling and message routing in *userspace*, backed by a *shared-memory* transport while leveraging Micro XRCE-DDS for typing, discovery, and routing. A lightweight Client on the controller side complements the host Agent implementation.

5.3 Methodology

5.3.1 Platform Overview

Figure fig. 5.1 outlines the HW/SW stack deployed in this work on a Xilinx ZCU102 FPGA board [122]. The PS hosts a quad Arm Cortex-A53 complex running Linux, where the XRCE-DDS *Agent* executes entirely in user space. On the PL side, the *ControlPULP* PCS RTL design is instantiated and runs a lightweight RTOS (FreeRTOS) [19] together with device drivers; it executes the Micro XRCE-DDS *Client*. Both subsystems access a common dynamic random access memory (DRAM) (4 GB) through the on-chip double data rate (DDR) memory controller; this region provides the substrate for the shared-memory transport used by Agent and Client.

At the middleware layer, the host side integrates RTPS services, while the controller side links the DDS-XRCE Client library. Message serializa-

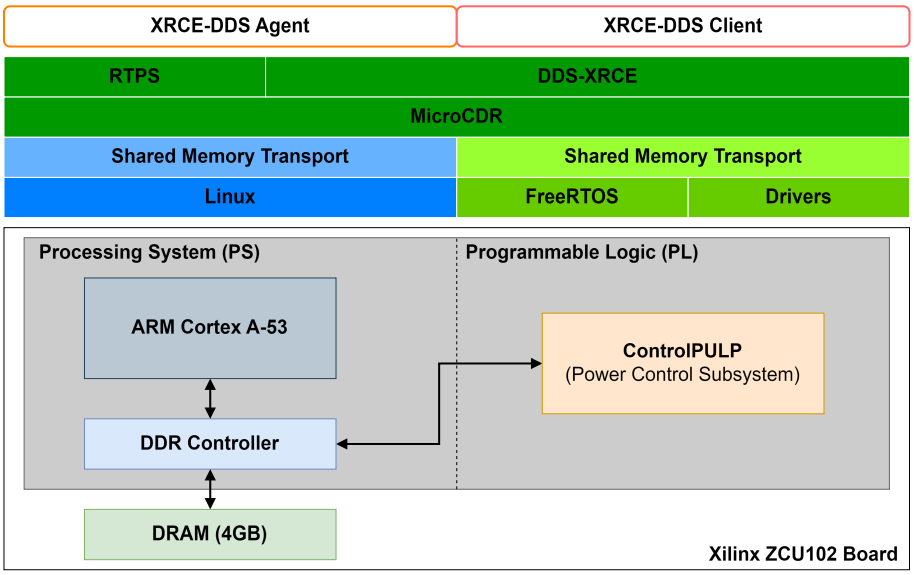


Figure 5.1: Micro XRCE-DDS implementation overview on Xilinx ZCU102 board.

tion/deserialization is performed with MicroCDR [45]. The shared-memory transport is implemented symmetrically on both endpoints and is exposed to the XRCE-DDS stack as a pluggable backend. Once the DRAM region is mapped, all message production/consumption, and telemetry handling reside in user space on the host and in firmware on the PCS; the kernel remains limited to address mapping and protection and does not mediate the data path.

5.3.2 Shared-Memory Transport

Shared-Memory Layout

The transport layer relies on a fixed shared-memory region accessible to both the host processor and the PCS; in this configuration, it resides in DRAM. As outlined in Figure fig. 5.2, the region is partitioned into two unidirectional paths, Agent→Client (A2C) and Client→Agent (C2A). Each path comprises a small set of *control fields* and a *payload buffer*. Two fields identify the message buffers for A2C and C2A directions, while two additional fields, `A2C_intr` and `C2A_intr`, act as *doorbells*: the producer raises the doorbell to signal availability of a new message; the consumer clears it upon successful consumption, thereby releasing the buffer and providing backpressure when the flag is already set. The `A2C_len` and `C2A_len` fields advertise the byte length of the message to be read in the corresponding buffer and are propagated to upper layers for framing.

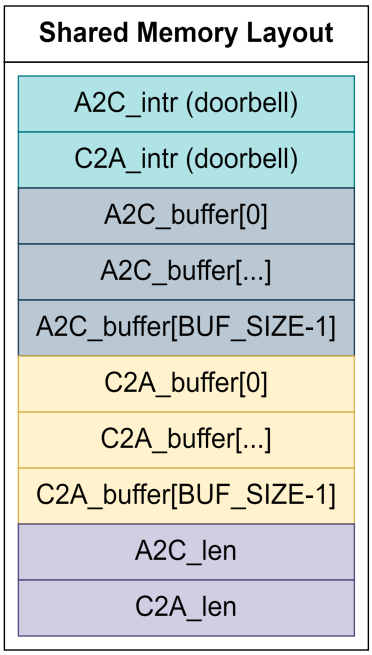


Figure 5.2: Shared Memory Layout for XRCE custom transport implementation.

Transport Software Backend

The transport is provided as a custom backend on both Agent and Client. Both endpoints access the same shared-memory organization at a common physical region, translated into their address spaces.

On the host (Agent), an initialization phase maps the region with `mmap()` function to cover the entire layout; kernel involvement is confined to mapping, while the XRCE stack remains in user space. Control fields are 32 bit, and each payload buffer is provisioned with 4 KiB. On the PCS, a C `struct` mirroring the layout in fig. 5.2 is bound to the shared region, enabling direct field access through pointers.

The producer–consumer interaction follows a strict sequence: *write payload* → *publish length* → *raise doorbell*; the consumer *reads length and payload* and then *clears the doorbell* to release the buffer. Timing parameters and measurement procedures are detailed in Section 5.4.

5.4 Evaluation

5.4.1 Software Setup

In realistic deployments, the XRCE Agent executes on a host with substantially higher compute capacity than the PCS; its workload depends on the number of

Clients and topics and is typically negligible compared to the host processors' computing capabilities. In this evaluation, then, we focus on XRCE Client read/write latency, message dispatching loop execution time, stack memory occupation, and shared memory occupation metrics.

In this setup, the Agent SW runs on the Arm Cortex-A53 of the ZCU102 with the only modification being the *custom shared-memory transport* backend described in section 5.3; no changes were made to Agent logic or RTPS bridging.

The XRCE–DDS Client is built with the following compile–time parameters:

- `UXR_CONFIG_CUSTOM_TRANSPORT_MTU=1024`: the transport Maximum Transmission Unit (MTU) is set to 1024 bytes. Application payloads are constrained not to exceed this value.
- `UXR_CONFIG_STREAM_HISTORY=2`: reliable streams keep at most two outstanding submessages before an acknowledgment is received, bounding buffering and stabilizing backpressure.

Middleware fragmentation is *disabled*. Each application sample fits within the transport MTU, mapping to a single shared–memory write/read without XRCE–level fragmentation or message reassembly.

The Client instantiates a single XRCE participant. A first *reliable* output stream is used exclusively for entity creation (participant, topics, publishers/subscribers); acknowledgments generated by the Agent are consumed via the shared–memory path. A second *best_effort* output stream carries topic data. *Best_effort* is adopted for the data path because delivery confirmation and backpressure are enforced at the transport layer by the shared–memory handshake (length fields and doorbells), avoiding per–sample retransmission in the middleware.

After initialization, a parametrized number of topics is created, $N_{\text{topics}} \in [1, 16]$. The application loop publishes 100 messages on each topic; each message has a fixed payload size of 32 bytes and a value that increases monotonically to enable end–to–end integrity checks at the Agent. For *best_effort* topic data, the producer–consumer protocol on shared memory (write payload → publish length → raise doorbell; consume → clear doorbell) provides flow control with at most one outstanding buffer per direction.

5.4.2 Firmware Execution Time

Execution timing was measured using the architectural performance counters available on ControlPULP, which provide cycle-accurate readings between two accesses to dedicated CSRs. All measurements were obtained at 20 MHz, corresponding to the frequency of the ControlPULP timer. The goal of this analysis is to characterize the latency of a single topic publication and to decompose it into its main execution phases.

For each message, measurement begins at the serialization stage, reported in table 5.1 as the *prepare* phase. The serialized payload is then dispatched through the XRCE–DDS session via `uxr_run_session_time()` function, which

handles both message transmission and potential reception of Agent responses. This function internally triggers one invocation of the transport *write* function and a sequence of *read* calls for polling; in the adopted best-effort configuration, acknowledgments are not generated, resulting in a continuous polling loop until timeout expiration.

Both *write* and *read* functions perform sequential byte copies to and from the shared-memory buffer, whose duration scales linearly with the message size. To isolate pure software overheads from waiting times, the duration of busy-wait phases—during which the Client polls for buffer availability—is measured separately and excluded where appropriate. Hence, we report the per-message execution breakdown, including: (i) message serialization (*prepare*); (ii) average per-byte memory-copy cost for write and read paths; and (iii) residual middleware bookkeeping excluding polling.

The serialization step takes approximately 49 μs, while the average per-byte copy time is 1.18 μs for writes and 1.28 μs for reads. For a 16 B message (4 B payload + 12 B XRCE framing), the write-side memory copy therefore accounts for roughly 18.9 μs. After removing the waiting and transfer phases, the residual software bookkeeping—including session management and function invocations—amounts to approximately 420 μs. Summing these contributions, the total active processing time per publication cycle is approximately 490 μs, representing the effective execution time of the Client firmware excluding idle polling.

The measured throughput is consistent in both transmission and reception: the effective data rate reaches about 0.8 MB/s, limited by the sequential memory-copy loops. These results confirm the symmetry of the shared-memory transport and the absence of software bottlenecks between the producer (Client) and consumer (Agent) sides.

Table 5.1: Timing characterization of Client execution phases (ControlPULP @ 20 MHz).

Measured phase	**Cycles**	**Time [μs]**
Message serialization (*prepare*)	980	49.0
Write loop (per byte)	23.6	1.18
Read loop (per byte)	25.6	1.28
Middleware bookkeeping (excl. polling)	8400	420.0

5.4.3 Stack Usage Measurement

Stack usage is measured via a watermarking technique: before execution, the entire stack region is filled with a known byte pattern; after the firmware execution, a linear scan from the stack base locates the first overwritten address, and the peak usage is computed as *stack_end* − *first_overwritten*. We evaluate the Client under two allocation strategies: (i) *dynamic (stack) allocation* of XRCE

Table 5.2: Peak stack usage and incremental cost per added topic.

N. Topics	Peak Usage [B]	Increment [B]	Bytes per added topic
1	2084	–	–
2	2356	+272	272.0
4	2900	+544	272.0
8	3972	+1072	268.0
16	6116	+2144	268.0

Extensible Markup Language (XML) descriptors (topics and datawriters) and per–topic bookkeeping, and (ii) *static allocation* of the same structures.

Results in Table table 5.2 show a monotonic increase under dynamic allocation of approximately +270 B per additional topic for $N_{\text{topics}} \in [1, 16]$. When the same descriptors are moved to static storage, the measured peak remains essentially constant (e.g., ≈1,828 B in our setup), independent of N_{topics}. This indicates that the observed growth is not intrinsic to the runtime publish path—which operates sequentially with a fixed per–iteration frame—but stems from placing per–topic descriptors on the stack; relocating these descriptors to static storage (or the heap) removes the linear component and yields a flat stack profile across topics.

5.4.4 Shared–Memory Occupation and Protocol Overhead

The shared-memory transport allocates two unidirectional buffers, each 4 KiB in size, used for bidirectional exchanges between the Client and the Agent. These buffers are sufficient to accommodate multiple MTU–sized frames (1024 B). Each endpoint interacts with the shared region through four 32–bit control fields (`len` and `doorbell` per direction), which provide lightweight synchronization but do not contribute to data–path overhead, as they are not part of the transmitted payload.

To quantify the actual space occupation, we analyzed the payloads captured by the Agent during Client execution. The log shows that a typical message for XRCE entity creation (participant, topic, publisher, and datawriter) occupies approximately 412 B in shared memory. The corresponding serialized content contains several XRCE submessages, each including both control and data sections, as shown in the Agent logs.

From these traces, the average *protocol overhead per submessage* is approximately 12–14 B, consistent with the XRCE–DDS specification. For telemetry samples of 32 B, this represents about 27% of the total message size, whereas for larger messages in the range of 400–1024 B the protocol overhead falls below 5%.

The transport layer itself contributes only the static synchronization fields (`A2C_len`, `C2A_len`, `A2C_intr`, `C2A_intr`), which remain persistent in memory and do not scale with message size.

As presented in table 5.3 for small messages, the fixed XRCE overhead

dominates the total byte count and reduces effective efficiency, as payload size increases, the same fixed 12–14 B cost becomes negligible.

Table 5.3: Protocol overhead as a function of payload size.

Message type	**Payload[B]**	**Overhead[B]**	**Total[B]**	**Overhead[%]**
Telemetry sample	32	12	44	27.3
Entity creation (single)	100	12	112	10.7
Entity creation batch	400	12	412	2.9
Large sample (MTU)	1024	14	1038	1.4

5.5 Conclusions

The evaluation presented in this chapter characterized the timing, memory, and communication behavior of the proposed user–space power–management interface based on the Micro XRCE-DDS middleware.

We first assessed the latency of a complete XRCE session loop on ControlPULP. The measurements show that the actual processing time of a single publication—encompassing message serialization, memory transfers, and middleware bookkeeping—amounts to approximately 490 µs. This value represents the effective execution time of the Client firmware, excluding idle polling periods imposed by session timeouts. In a realistic silicon implementation of ControlPULP, where clock frequencies are up to 25× higher than in the FPGA implementation, the same operations would execute within only a few tens of microseconds, ensuring ample timing margin for sub-millisecond control cycles.

We then evaluated peak stack usage during best-effort communication. Stack occupation grows linearly with the number of created XRCE entities (topics and datawriters), with an incremental cost of roughly 270 B per topic, while reaching about 1.8 KiB when static allocation is adopted. No additional overhead was observed at runtime: once entities are instantiated, the per–iteration stack frame remains constant, with no dynamic structures allocated along the session loop.

Finally, we analyzed shared-memory usage for message exchange. For unfragmented messages, the largest entity–creation message occupies approximately 412 B, while typical telemetry samples are limited by the 1 KiB MTU. The main memory consumers are thus the XRCE entity–creation packets, whereas steady–state topic communication remains lightweight and predictable.

Overall, the obtained results demonstrate that the proposed Micro XRCE-DDS integration enables low-latency, low-footprint communication between the host and the controller, in which the kernel is only involved in the initial shared-memory mapping and does not host protocol logic. Thanks to the user-space transport backend, this architecture allows power-management policies to be implemented directly on top of the Agent stack, enabling policy daemons or user-level services to exchange telemetry and performance requests without

kernel-resident protocol handling and without requiring kernel/driver changes. Given the bounded stack and shared-memory requirements on the controller side, the same design naturally scales to multi–PCS configurations, where multiple XRCE Client instances—each corresponding to a distinct chiplet controller—communicate with a shared Agent through replicated shared-memory regions within the global DRAM.

Chapter 6

Compact Thermal and Power Models for Many-core chiplet-based Architectures

6.1 Introduction

The increasing complexity of HPC workloads and AI algorithms, such as Large Language Model (LLM), has driven a continuous demand for higher performance in modern systems. This has resulted in a widening gap between the growth in computational performance and the slower improvements in energy efficiency, leading to a steady rise in power consumption [92].

In processor design, the end of constant-power scaling, combined with the relentless pursuit of performance, has shifted the paradigm from single-core enhancements to the integration of many cores within a single package [46]. However, monolithic scaling is ultimately constrained by semiconductor manufacturing limits on die size and maximum core count, motivating the adoption of chiplet-based architectures.

Over the past decade, chiplet-based designs have rapidly evolved. In the HPC domain, processors have scaled from four chiplets configurations, such as Intel Sapphire Rapids [35], to 13-chiplet designs, such as AMD EPYC Genoa [4]; in AI accelerators, systems like Intel Ponte Vecchio[60] integrate up to 47 tiles. With individual chiplets already close to the reticle size limit, further performance scaling relies on increasing the number of chiplets per package directly leading to a larger package area, higher total power consumption, and consequently more severe thermal challenges.

As package power and chiplet integration continue to scale, thermal behavior emerges as a primary performance constraint. A single overheated core can

trigger package-level thermal protection mechanisms, forcing a global frequency reduction and penalizing the performance of all workloads running on the processor. Elevated temperatures also exacerbate leakage power, tightening power and current budgets and inducing successive layers of throttling, including power, current, and thermal capping, with significant losses in both performance and energy efficiency. Existing hardware protection mechanisms are predominantly threshold-based and therefore inherently reactive, as intervention occurs only after a hotspot has already formed. To guarantee safe operation across diverse workloads and cooling conditions, such mechanisms rely on conservative guardbands that substantially limit the attainable performance and are often ineffective in advanced cooling environments, such as liquid-cooled systems [80].

Predictive thermal information enables proactive control mechanisms. Thermal-aware schedulers can exploit compact thermal models to estimate the impact of alternative task placements and avoid hotspot formation, while power management controllers can rely on the same models to implement advanced control strategies, such as Model Predictive Control (MPC), that dynamically regulate voltage, frequency, and power limits of PEs to improve the overall system performance.

In this context, accurate thermal characterization and modeling are therefore critical. Two main approaches can be identified. Simulation-driven models rely on knowledge of the processor floorplan and detailed thermal parameters. However, such information is typically unavailable for commercial platforms and, even if accessible, depends strongly on the deployment environment and device aging [74], [120]. Data-driven empirical models, on the other hand, extract thermal behavior directly from measurements, avoiding dependence on physical design details and enabling direct applicability to real systems [93], [26].

Prior work [40], [41], [93], [26] has focused on monolithic processors with relatively few cores, where thermal interactions could always be analyzed under full-core activation. In those cases, thermal models were also leveraged to reconstruct the processor floorplan, a task made tractable by the limited number of cores and the relatively simple die geometry, which avoided the need for complex allocation formulations. This approach is infeasible in modern many-core chiplet-based systems, where scalability constraints and the distributed nature of chiplets introduce a far more challenging characterization problem.

To address these limitations, the main goal of this chapter is to bridge this gap by proposing a scalable, data-driven framework capable of enabling accurate modeling and floorplan reconstruction without design-time knowledge.

- **Distributed thermal modeling and intra-chiplet inference.** We propose a novel distributed Multiple-Input Single-Output AutoRegressive model with eXogenous inputs (MISO-ARX) identification methodology that scales to chiplet-based processors and extracts the thermal coupling between processor cores. As an enabling component, we also introduce a hierarchical per-core power estimation model derived from processor-level measurements, which provides the power traces required for thermal identification on commercial platforms.

- **Processor floorplan inference and validation.** We introduce a floorplan inference algorithm capable of reconstructing the complete layout of a commercial many-core chiplet-based processor, including both chiplet partitioning and fine-grained intra-chiplet core placement, without access to vendor-disclosed floorplan information. By aligning empirically estimated thermal coupling strengths with spatial adjacency constraints, the proposed formulation yields a physically consistent topology.

  The inferred layout is validated by comparing thermal prediction accuracy under different spatial input configurations. Models exploiting only Manhattan-1 neighbors achieve a median MSE of $3.988\,^{\circ}\mathrm{C}^2$, corresponding to a 7.4% reduction with respect to single-core models, while the full-information configuration achieves a median MSE of $3.756\,^{\circ}\mathrm{C}^2$ (12.8% reduction). These results demonstrate that most of the predictive information is spatially local and that the reconstructed adjacency meaningfully reflects the dominant thermal interaction structure of the processor.

6.2 Related Work

6.2.1 Thermal Modeling

Simulation-Driven Thermal Modeling

Several approaches model processor temperatures through simulation frameworks that rely on detailed knowledge of the floorplan, material properties, and package structure. Notable examples include thermal-aware placement and analysis tools for multi-chiplet systems [120], as well as compact transient thermal simulators [123]. These approaches can capture heat spreading with fine-grained accuracy, often achieving mean absolute errors as low as 1.2°C. They consistently report that thermal coupling is significantly stronger within each chiplet than across chiplets, with inter-chiplet transfer depending heavily on interposer spacing and packaging materials. However, they require inputs such as per-block power dissipation and geometric details that are seldom available for industrial processors. Moreover, their fidelity is sensitive to manufacturing tolerances, device aging, and deployment conditions (e.g., cooling setup, ambient temperature). As a result, simulation-driven models are primarily used at design time and are not suitable for runtime thermal-aware management.

Empirical Thermal Modeling

Empirical identification overcomes these limitations by deriving thermal models directly from measurements on the deployed system. This enables extracting compact, physically consistent models, and in some cases, even recovering the processor floorplan, without requiring internal design data. Early works [26] applied optimization-based techniques to characterize a quad-core processor. The approach in [41], [40] extended this line through MISO-ARX+noise models with bias compensation, which yield stable dynamics under quantization noise

and environmental disturbances. Subsequent studies validated these models in production HPC settings with controlled excitation [40]. However, all these efforts focused on monolithic processors with few cores.

6.2.2 Floorplan Inference From Measurements

Beyond temperature prediction, empirical studies also investigated floorplan inference from thermal couplings. As shown in [40], the relative placement of cores and uncore blocks on a small monolithic processor can be recovered by visually inspecting the coefficients derived from the identified thermal model, since the limited number of components and the regular die geometry enable a straightforward matching between coupling strengths and physical adjacency. Other approaches infer spatial information from on-chip communication behavior. For example, in [75] directional traffic performance counters of the on-chip mesh interconnect are exploited to map cores and memory controllers to their physical die locations. In contrast to the visual inspection strategy adopted in [40], modern many-core multichiplet processors exhibit a much more complex interaction structure, and the weak geometric constraints make layout inference significantly harder. Moreover, in our experimental setup, mesh traffic performance counters required by communication-based techniques are not available. As a result, an explicit quadratic assignment formulation is required to recover a floorplan consistent with the observed thermal couplings.

6.2.3 Power Modeling From Performance Counters

A complementary line of work estimates package power using architectural performance counters, typically via linear or regularized regression models [93]. These models are essential enablers for thermal identification when direct per-core power measurements are unavailable. Prior work, however, has largely targeted monolithic CPUs and general-purpose HPC workloads. Extending counter selection and attribution mechanisms to many-core chiplet-based systems, while retaining per-core resolution, remains an open challenge.

6.3 Methodology

The proposed thermal and power model learning infrastructure is illustrated in Fig. 6.1. The system is organized into five primary functional layers: Workloads, Monitoring and Processing, Power Modeling, Thermal Modeling, and Processor Floorplan Inference, detailed in the following subsections.

6.3.1 Workloads

Following the approach in [26], we employ synthetic *power viruses* [32] that drive selected cores to 100% utilization when activated. The activation patterns of these viruses are controlled through input traces generated offline. Two types

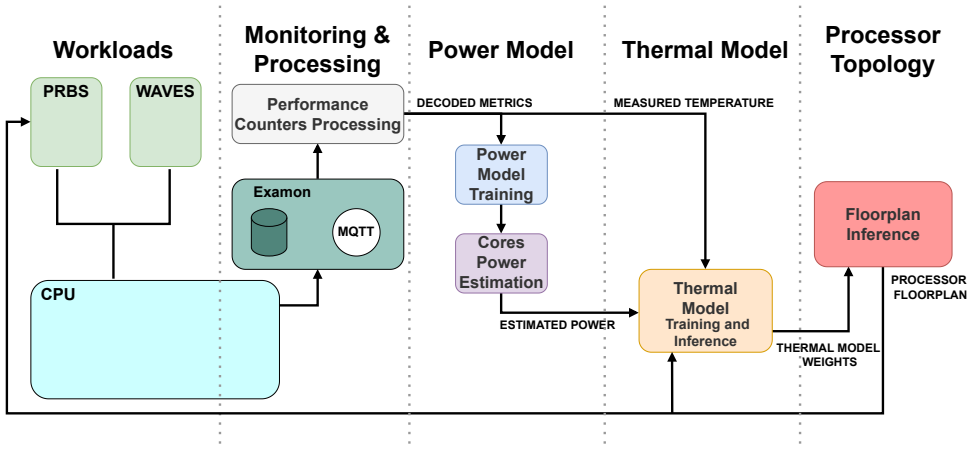


Figure 6.1: Monitoring and processing framework architecture overview.

of traces are employed in our experiments: *Pseudo-Random Binary Sequence (PRBS)*, which modulate the activation of power viruses across cores and thereby provide the excitation diversity required for system identification; and *STEPS*, consisting of sustained full-load activation on selected cores for fixed-duration intervals, used to analyze power and thermal behavior under constant workload conditions. Once a workload type is selected, each trace corresponds to one execution *round*, and each round has its target cores for the specific kind of workload.

6.3.2 Monitoring and processing

The monitoring and processing framework is organized into two complementary components: *Performance Monitoring* presented in fig. 6.2, responsible for the acquisition and publication of architectural metrics, and *Data Processing*, which handles the decoding, transformation, and temporal alignment of the collected data for subsequent power and thermal model estimation.

Performance Monitoring

During each stress workload execution, the framework collects a comprehensive set of architectural performance counters. In our setup, the selected counters correspond to per-core frequency and temperature, core and package C-states, uncore frequency, package temperature, and Top-Down microarchitectural metrics. These measurements are logged locally and subsequently processed to decode, aggregate, and align them with workload traces, providing consistent inputs for the power and thermal models. The monitoring subsystem is orchestrated by the `PMU_PUB` script [43], which acts as the main data acquisition engine within the framework. `PMU_PUB` operates periodically, with a configurable sampling

period, and queries the available hardware performance counters through two complementary tools: `perf_event` and `sensor_lib`. The `perf_event` subsystem is the standard Linux kernel interface for configuring and reading Performance Monitoring Units (PMUs). It allows user-space applications to monitor a wide range of hardware events, such as instructions retired, cache misses, and frequency states, through the kernel-mediated configuration of the underlying PMU registers. Conversely, `sensor_lib` is a lightweight library developed within this framework to access performance data exposed as architectural MSRs. The library encapsulates the necessary access mechanisms to read these registers directly, enabling precise retrieval of per-core metrics such as temperature and frequency, complementing the data acquired via `perf_event`. All collected samples are aggregated and published synchronously through the MQTT [84] protocol to a local time-series database managed by **Examon** [47]. This modular publish/subscribe infrastructure decouples the monitoring layer from higher-level analytics and visualization components.

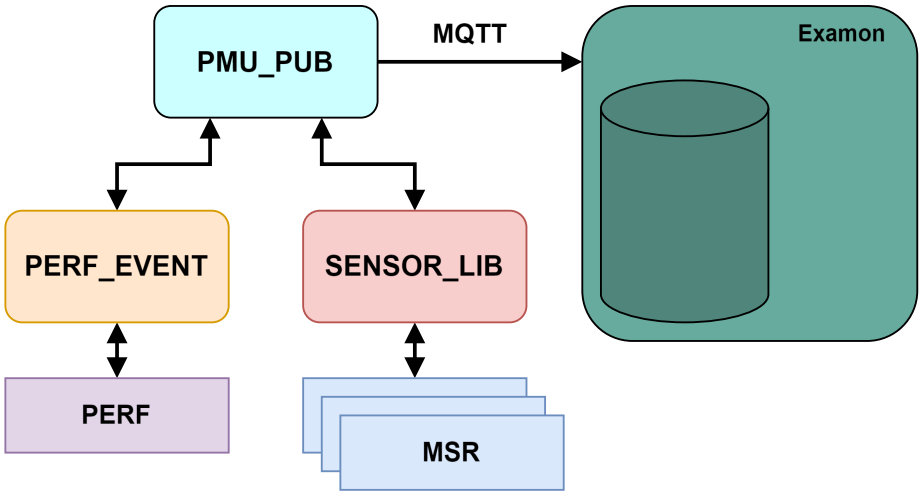


Figure 6.2: Performance monitoring software architecture overview.

Data Processing Pipeline

The data processing stage of the framework [42] is responsible for decoding and transforming the raw metrics collected from architectural performance counters into logical and physical quantities suitable for power and thermal modeling. This stage operates as a configurable processing loop that continuously parses and elaborates the locally stored counter data. This processing engine takes as input two main elements: the local database containing the raw performance counter logs, and a configuration file that defines how each metric should be processed. The configuration file specifies, for each counter, the metric to be extracted, the type of operation to perform, and additional parameters such as

bit width, handling rules for missing values, and whether a metric should be delivered as an output. This structure allows the pipeline to be easily adapted to different target platforms and monitoring setups without modifying the core code. The elaborated data are then organized into structured `pandas`[77] dataframes, separated by hierarchy level to facilitate metric-specific processing. Two main categories of metrics are distinguished: *core-level* and *package-level*. Core-level metrics refer to quantities associated with individual processing cores (e.g., per-core frequency and temperature); package-level metrics, instead, represent quantities aggregated at the processor package level (e.g., package temperature, uncore frequency). Once all metrics have been processed, the engine aligns the resulting dataframes to a common time index to ensure synchronization across all signals.

6.3.3 Thermal Model

We model core temperatures using a compact MISO-ARX+noise structure proposed in [41], which yields stable and physically consistent dynamics even under quantized and noisy sensor data. Following [41], we fix model order to n=2 and sampling period T_s=0.5 s. For each target core, the model predicts the next temperature sample from past temperature measurements and per-core power traces. Model parameters are estimated via bias-compensated least squares [41], using measured temperature samples and the corresponding per-core power traces. Excitation is ensured through PRBS modulated power-virus workload, while one-step-ahead prediction relies on a state-space realization and a Kalman filter [94].

6.3.4 Power Model

The thermal identification strategy detailed in Sec. 6.3.3 relies on per-core power traces as inputs. Since such measurements are not available on commercial processors, we derive them through a hierarchical modeling approach. First, we derive a processor-level model using the `ERG_PKG` counter as the target. This linear regression model is optimized via Stochastic Gradient Descent (SGD) and utilizes both core metrics (e.g., average temperature, frequency, C0 residency) and uncore metrics (e.g., uncore frequency). Second, we derive the per-core power model by redistributing the global coefficients. Specifically, the weights associated with core-dependent metrics are normalized by the number of cores in a processor. These normalized weights are then applied to the individual performance counters of each core to estimate its specific power consumption, while uncore metrics are treated as a shared background contribution. For evaluation, the dataset was partitioned into training and test sets using a 60/40 split, and both the input features and the target variable were standardized using a *StandardScaler* to ensure consistent scaling.

6.3.5 Processor Floorplan Inference

In this chapter, the processor floorplan is defined as the mapping between each core identifier and its physical position in the processor layout. Although the detailed physical floorplan of the Intel Sapphire Rapids processor [35] is not publicly disclosed, some coarse-grain architectural information is available from vendor documentation.

Specifically, the processor integrates four chiplets, each hosting a fixed number of cores, one Memory Controller (MC), and one permanently disabled core reserved for yield and redundancy. Moreover, cores are known to be grouped into chiplets, while their exact spatial arrangement within each chiplet is not documented. This information does not determine the physical layout of the cores, but it defines a set of structural constraints that any feasible floorplan must satisfy. In our approach, these constraints are used to restrict the solution space, while the actual placement of cores within each chiplet and the relative positioning of the chiplets are inferred from the measured thermal coupling strengths.

Given the thermal coupling information extracted in Sec. 6.3.3, our goal is to infer a plausible physical floorplan of the processor and reduce the complexity of per-core thermal models by restricting interactions to spatially local neighborhoods. To this end, we formulate the problem as a Quadratic Assignment Problem (QAP).

Problem Formulation. The processor integrates 56 active cores distributed across four identical chiplets. Each chiplet, indexed by $k \in \{0, 1, 2, 3\}$, can be represented as a 4×4 grid of 16 slots: one slot hosts the MC, whose position is known, while 15 slots are reserved for cores, among which one is permanently disabled. Thus, for each chiplet, $n_c = 14$ cores are effectively characterized, and $n_s = 15$ slots are available, excluding the fixed MC position. Formally, for each chiplet k, let $\mathcal{C}_k = \{1, \ldots, n_c\}$ denote the set of active cores and $\mathcal{S}_k = \{1, \ldots, n_s\}$ the set of slot positions in the local 4×4 grid. We introduce binary assignment variables:

$$x_{c,s}^{(k)} \in \{0, 1\}, \quad \forall c \in \mathcal{C}_k, \forall s \in \mathcal{S}_k,$$

where $x_{c,s}^{(k)} = 1$ if core c is assigned to slot s.

The inference problem is formulated as a QAP:

$$\max_{x} \sum_{i,m \in \mathcal{C}_k} \sum_{s,l \in \mathcal{S}_k} \left(A_{im}^{(k)}\right)^2 N_{sl}^{(k,p)} \, x_{i,l}^{(k)} \, x_{m,s}^{(k)} \; + \; \eta \sum_{c \in \mathcal{C}_k} x_{c,\, s_{\text{corner}}^{(k)}}^{(k)},$$

where:

- $A^{(k)} \in \mathbb{R}^{n_c \times n_c}$ is the thermal coupling matrix, derived from data-driven estimation (Sec. 6.4.3), whose entries $A_{i,m}^{(k)}$ quantify the strength of the thermal interaction between cores i and m.

- $N^{(k,p)} \in \mathbb{R}^{n_s \times n_s}$ encodes the geometric adjacency between slots s and l under orientation pattern $p \in \{\text{base}, \text{rotated}\}$.
- η is a small corner penalty term that discourages excessive core clustering in corner positions.

The objective promotes an assignment where cores with strong thermal couplings are placed in geometrically close positions according to $N^{(k,p)}$. The optimization is subject to a set of constraints ensuring a valid and physically consistent floorplan assignment. Specifically:

- **Unique assignment:** each core must be assigned to exactly one slot;
- **Slot capacity:** each slot can host at most one core;
- **MC exclusion:** MC slots are prohibited and cannot be assigned to any core.

Formally, for each chiplet k, these constraints can be written as:

$$\sum_{s \in \mathcal{S}_k} x_{c,s}^{(k)} = 1, \qquad \forall c \in \mathcal{C}_k, \quad \text{(unique assignment)} \tag{6.1}$$

$$\sum_{c \in \mathcal{C}_k} x_{c,s}^{(k)} \leq 1, \qquad \forall s \in \mathcal{S}_k, \quad \text{(slot capacity)} \tag{6.2}$$

$$x_{c,s}^{(k)} = 0, \qquad \forall c \in \mathcal{C}_k,\ s \in \mathcal{S}_k^{(\text{MC})}, \quad \text{(MC exclusion)} \tag{6.3}$$

where $\mathcal{S}_k^{(\text{MC})} \subset \mathcal{S}_k$ denotes the subset of slots reserved for the MC and therefore excluded from assignment.

Model Refinements. To improve solver robustness and accelerate convergence, several refinements are introduced:

1. **Adjacency filtering.** For each core, only the strongest thermal couplings are retained by selecting the top-N entries per row of $A^{(k)}$. The chosen couplings are then attenuated through a scaling function (e.g., rank-based or exponential decay). This attenuation prevents the formation of overly rigid clusters around highly coupled cores, which would otherwise dominate the objective function and penalize the placement flexibility of the remaining ones. By softening secondary interactions, the solver explores a smoother optimization landscape and avoids local optima driven by a few dominant couplings.
2. **Corner penalty.** Without additional constraints, the optimization tends to favor configurations where corner slots are assigned to disabled cores, while active cores are relocated toward the chiplet center, thus increasing the objective value. This effect arises not only because a corner slot has fewer neighboring positions and therefore contributes less to the coupling

term, but also because the active cores located near the boundary pull other active cores inward while pushing the inactive one toward the corner, since the latter does not improve the objective. Moreover, disabled cores are excluded from the adjacency matrix $A^{(k)}$, as they are not visible to the operating system and cannot execute workloads or expose performance counters. Consequently, no explicit term in the objective function accounts for their placement, and the solver naturally drives them toward peripheral positions. To counteract this behavior, a small penalty term η is added to the objective function, discouraging the allocation of active cores to corner slots and reducing excessive peripheral bias.

Solver Implementation. All quadratic assignment subproblems are solved using the `Gurobi` [52] optimizer, configured with a non-convex quadratic objective and solution pool exploration enabled. The solver is employed both for the chiplet-level floorplanning problems and for the reduced inter-chiplet placement problem. For the intra-chiplet stage, each chiplet is optimized independently, generating multiple candidate allocations. The solver explores combinations of random seeds and corner penalty values to enrich the solution pool and mitigate local optima. Each solution is stored together with its objective value and provenance metadata (e.g., random seed and penalty setting) for subsequent stability analysis.

Inter-chiplet placement. The global thermal coupling matrix is first coarsened by aggregating coefficients into 14×14 blocks, one per chiplet, and nulling out the diagonal terms. This yields a reduced 4×4 matrix that captures the average thermal interactions among chiplets. Solving a small assignment problem on this reduced representation provides a relative 2×2 placement of the chiplets. Due to symmetry in the MC orientation, two equivalent global orientations are retained for subsequent stages.

Intra-chiplet floorplanning. Each chiplet is then optimized independently using its local thermal coupling matrix $A^{(k)}$, with memory-controller slots masked out. The QAP formulation described above is applied at the chiplet level to determine a consistent assignment of cores to slots within each tile.

Global integration. Since the intra-chiplet optimization and the chiplet-level orientation may produce multiple candidate configurations, a global integration step is required. At this stage, partial solutions are combined into full-package layouts and filtered through a consistency analysis, which identifies stable core-to-slot assignments across multiple runs. The resulting candidate floorplans are then evaluated using the full-resolution thermal coupling matrix and the global neighborhood structure to select a configuration that best aligns thermal interaction strengths with spatial proximity.

6.4 Evaluation

6.4.1 Experimental Setup

Experiments were performed on a single-node server hosting two Intel Xeon Platinum 8480+ CPUs (Sapphire Rapids) [35], with only one processor enabled per run. The single CPU integrates 56 physical cores with Hyper-Threading disabled. The node features 256 GB of DDR5 Random Access Memory (RAM) at 4800 MHz on a Supermicro X13DET-B motherboard, running Rocky Linux 9.5 with kernel 5.14. Cooling is provided by two rack-mounted air systems operating at a fixed fan speed of ∼15,000 RPM during all tests.

6.4.2 Power Model

Per-chiplet Power Consumption Since the per-core power model presented in Sec. 6.3.4 is obtained by averaging the coefficients of a linear model, it is essential to validate the assumption that cores within different chiplets exhibit the same power consumption under identical workloads. The actual distribution of cores across chiplets is not disclosed by the vendor; therefore, at this stage we rely on the representation provided by `numactl` Linux command, which groups cores into four Non-Uniform Memory Access (NUMA) domains potentially corresponding to the physical chiplets. If any imbalance in chiplet power consumption were present, it would emerge from the experimental measurements.

The validation campaign is divided into two parts. In the first part, we execute the *STEPS* workload in 20 *rounds*, grouped as five *rounds* per chiplet. Within each *round*, we activate $N \in 1, 2, 4, 8, 14$ randomly selected cores from the same chiplet. This test evaluates whether different chiplets consume systematically different amounts of power for the same number of active cores.

In the second part, we analyze possible uncore power offsets by distributing the same number of active cores across multiple chiplets. Specifically, we consider 12 configurations where 4, 8, or 14 cores are activated and spread across 2, 3, or 4 chiplets. This setup allows us to assess whether the activation of the first core in a chiplet introduces an additional uncore power contribution.

Each workload execution is structured into three phases: an idle period of 5 s, an active phase of 60 s, and a post-activity idle period of 15 s. All configurations are repeated 10 times to enable statistical averaging of the results. Fig. 6.3 reports the average package power measured during the active phases when activating $N \in \{1, 2, 4, 8, 14\}$ cores *within a single chiplet at a time*, repeating the experiment independently for each chiplet. No consistent or systematic differences emerge among chiplets for the same number of active cores. Similarly, distributing active cores across multiple chiplets does not significantly alter average package power consumption, provided that the total number of active cores remains constant. These results hold for the *PRBS* workload, which, like the *STEPS* workload, is CPU-bound.

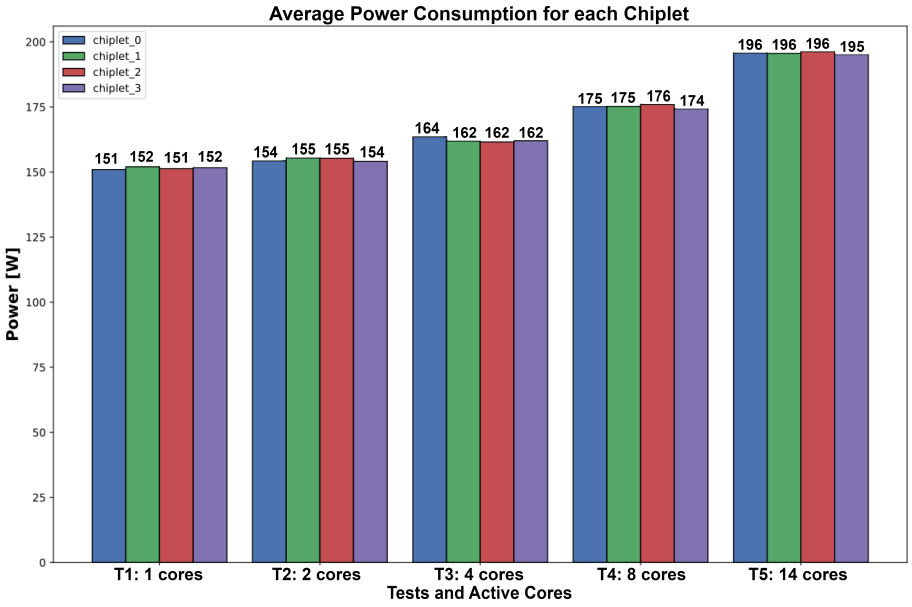


Figure 6.3: Average processor power consumption over different configurations of active cores for each chiplet.

Idle Power Consumption To characterize idle power, we concatenate the idle samples collected across both experimental campaigns. For each round, we extract the 5-second interval preceding the active phase and compute the average CPU power and average temperature. This produces a dataset of idle operating points across a range of thermal conditions.

Fig. 6.4 shows the linear regression of idle power as a function of average CPU temperature. In the observed range of 49.5–52.5°C, idle power exhibits a clear dependence on temperature. The fitted linear model is

$$P_{\text{idle}} = 0.33 \cdot T + 130,$$

with a coefficient of determination $R^2 = 0.59$. This model provides the reference P_{idle} used in subsequent analyses.

Metrics selection To identify a robust set of input metrics for the power model, we began by considering the feature set proposed in [93]. In our case, however, preliminary tests revealed that only a subset of these metrics exhibited non-negligible coefficients when fitting the processor-level power model. This can be explained by the different nature of our workloads: while [93] targeted realistic HPC applications, here we focus on binary workloads that exercise the processor in a more constrained way. Following the methodology suggested in [93], we therefore reinitialized our feature set using only the product of `C0 residency` and `core frequency` as a baseline, and progressively explored the

contribution of the remaining metrics that had shown larger coefficients in the preliminary analysis.

While this configuration allowed the model to capture average power consumption, it failed to reproduce several dynamic effects observed in practice. In particular, the model was unable to explain (i) the gradual drifts in average power during both active and idle phases, and (ii) sharp spikes that typically occur during high-load intervals.

To address this limitation, we added per-core temperature as an additional input feature. With temperature included, the model successfully captured the slow variations in power during sustained workload and idle periods, confirming the significant role of thermal feedback in shaping dynamic power consumption.

Analysis of the performance counters further revealed that the unexplained spikes coincided with variations in uncore activity. Consequently, we extended the feature set with the uncore frequency counter (`uclk`).

The same model was subsequently evaluated on the *PRBS* workload, obtaining an Root Mean Square Error (RMSE) of 2.464 W over 500 samples at a sampling period of $T_s = 0.5$ s.

The final set of selected metrics and their regression coefficients are summarized in Table 6.1.

Table 6.1: Selected input metrics and regression coefficients for the power model.

Metric	**Coefficient**
temp_core	0.218433
freq*C0_core	1.517517
uclk_cpu	1.120655

6.4.3 Thermal Model

Applying the state-of-the-art approach [41],[26] directly to the full package would result in a system identification problem with 56 cores and 6384 parameters. Given the large number of cores in the target processor, the identification performance of the adopted MISO-ARX thermal model may depend on the number of exogenous inputs used for identification, and, more critically in this context, on which cores are selected as inputs, since thermal interactions are inherently distance-dependent. For this reason, the identified models are exploited to derive a spatial coupling map aimed at reconstructing the processor floorplan.

A key aspect of this process is the design of the identification dataset, defined by the selection of which cores execute the excitation workload in each round and by how such selections are organized across rounds. Since the goal is to emphasize local thermal interactions, the dataset is constructed so that, over

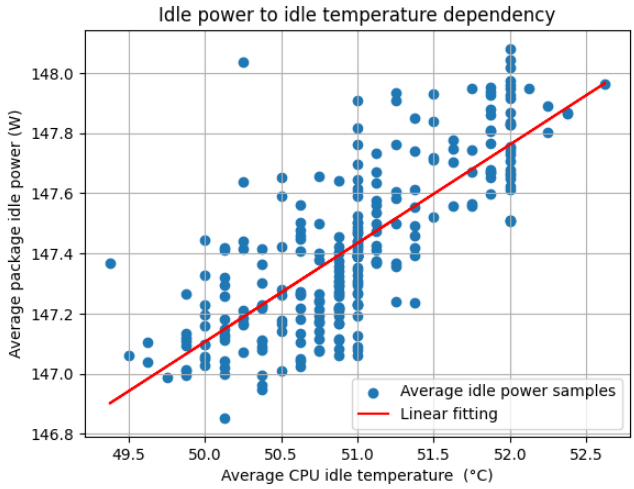


Figure 6.4: Idle power vs. temperature dependency.

the entire identification campaign, every pair of cores in the processor executes the workload in the same round at least once.

Using only a very small number of simultaneously executing cores may lead to weakly informative temperature traces, as the generated heat is rapidly spread over neighboring regions and local interactions become less observable. Conversely, involving a large number of cores in the same round injects a large and spatially distributed thermal excitation into the system, which reduces the contrast between local and non-local thermal interactions and makes individual pairwise couplings harder to distinguish, ultimately limiting the usefulness of the resulting coupling map for floorplan reconstruction. To balance these effects, each round is defined by a subset of 14 randomly selected cores executing the *PRBS* workload for identification. In total, we performed 660 rounds.

The thermal coefficients obtained from all rounds were then averaged, and the Euclidean norm of the exogenous terms was computed for each core to build a coupling map. Formally, for each pair of cores i, m, the coupling strength is defined as

$$A_{i,m} = \frac{1}{|\mathcal{R}_{i,m}|} \sum_{r \in \mathcal{R}_{i,m}} \left\| b_{i \leftarrow m}^{(r)} \right\|_2 ,$$

where $b_{i \leftarrow m}^{(r)}$ are the exogenous MISO-ARX+noise coefficients estimated in round r, and $\mathcal{R}_{i,m}$ is the set of rounds where both i and m are active. The resulting matrix A represents the global coupling map.

As shown in Fig. 6.5, this map highlights strong thermal interactions within each chiplet and weaker ones across chiplets, thus validating the assumed correspondence between NUMA domains and chiplet organization.

The predictive performance of the identified thermal model under full-chip excitation—i.e., with all 56 cores simultaneously active—is evaluated separately

in Sec. 6.4.4, where different input configurations are compared.

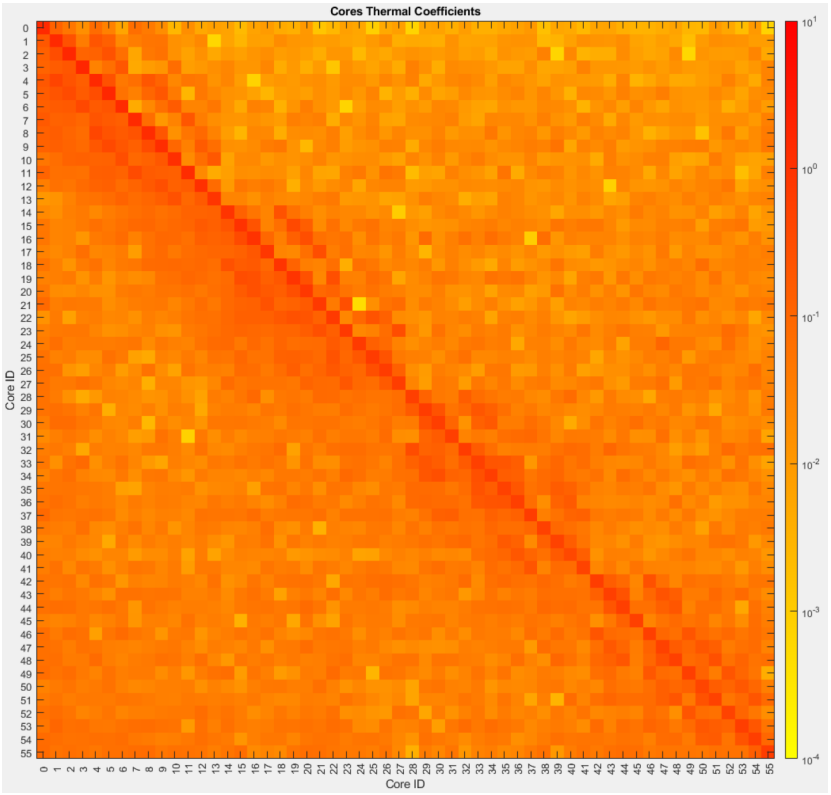


Figure 6.5: Global thermal coupling coefficient map in logarithmic scale. Each row index identifies the core for which the thermal model is derived, whereas each column index denotes the core whose exogenous thermal coefficients contribute to the modeled temperature, producing the value reported in the heatmap.

6.4.4 Processor Floorplan Inference

Resulting floorplan. Applying the inference procedure described in Sec. 6.3.5 to the global thermal coupling matrix yields the processor floorplan shown in Fig. 6.6. The figure reports the four inferred chiplets, labeled C0–C3, together with the allocation of cores within each chiplet, identified by their core IDs. In addition to active cores, the layout also shows the positions of the redundant cores (marked as "R") and the memory controllers ("MC"), whose presence and count per chiplet are known from public documentation and were therefore imposed as structural constraints in the optimization process. While the positions of

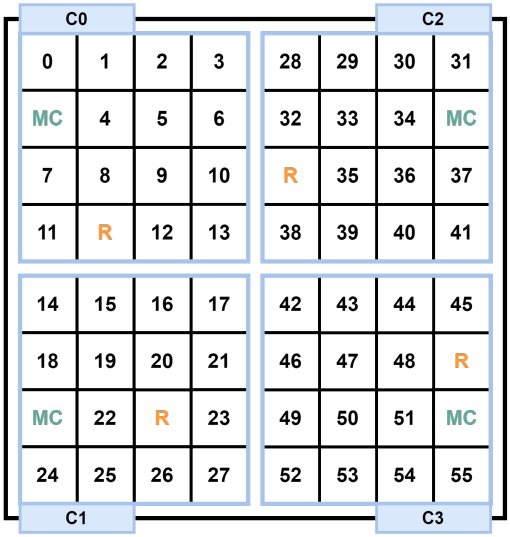


Figure 6.6: Inferred processor floorplan showing the four chiplets (C0–C3) and the mapping of core identifiers to physical positions. Core IDs label active cores, "MC" indicates memory controllers, and "R" denotes redundant cores.

the memory controllers are fixed, the locations of both active and redundant cores are entirely determined by the proposed inference algorithm. The inferred allocation exhibits a regular row-wise ordering of core identifiers within each chiplet, with the linear core numbering disrupted only by the presence of memory controllers and redundant cores. This organization is qualitatively consistent with publicly documented architectural observations for Intel Xeon Max processors, which are Sapphire Rapids variants sharing the same four-chiplet, up to 56-core structure [76], and supports the physical plausibility of the reconstructed layout.

Validation. The inferred floorplan is validated by comparing the predictive accuracy of thermal models identified under different input configurations derived from the reconstructed spatial topology in Fig. 6.6.

Thermal identification is performed under full-chip *PRBS* excitation, in which all cores execute the workload simultaneously. For each core, the available power and temperature traces are split into 60% identification data and 40% test data. One-step-ahead prediction is evaluated on the test portion using the MISO-ARX+noise model described in Sec. 6.3.3, and the MSE is computed for each core.

Four input configurations are considered, corresponding to different subsets of exogenous core power traces: (i) *all*, including the target core together with all remaining cores; (ii) *cross*, including the target core and its Manhattan-1 neighbors according to the inferred floorplan; (iii) *single*, including just the target

core; (iv) *corners*, including the target core and the four chiplet corner cores.

The resulting prediction error distributions across cores are summarized in Fig. 6.7. The *all* configuration provides a reference lower bound on the achievable prediction error, as it exploits the maximum available exogenous information. It achieves a median MSE of $3.756\,^{\circ}\mathrm{C}^2$, corresponding to a 12.8% reduction with respect to the *single* configuration (median MSE $4.306\,^{\circ}\mathrm{C}^2$). The *single* configuration serves as a baseline and already achieves relatively low prediction errors, indicating that a significant portion of the thermal dynamics is captured by the autoregressive component and local power dissipation. Incorporating spatially adjacent cores through the *cross* configuration consistently reduces the MSE to a median value of $3.988\,^{\circ}\mathrm{C}^2$, corresponding to a 7.4% improvement over *single*. Importantly, *cross* and *corners* use input sets of equal cardinality. However, *corners* achieves only a marginal improvement (median $4.189\,^{\circ}\mathrm{C}^2$, i.e., 2.7% over *single*). The systematic advantage of *cross* therefore cannot be attributed to model dimensionality, but to spatial coherence with the inferred topology.

While the *all* configuration achieves the lowest overall error, the improvement over *cross* is moderate, indicating that most of the predictive information is already captured by spatially local neighbors. Overall, the systematic ranking

$$\mathit{all} \;\leq\; \mathit{cross} \;<\; \mathit{corners} \approx \mathit{single}$$

provides empirical validation of the inferred floorplan, demonstrating that the reconstructed spatial adjacency meaningfully reflects the dominant thermal interaction structure of the processor. Although the adopted model structure follows established state-of-the-art identification techniques, achieving this level of accuracy on a 56-core multi-chiplet commercial processor under full-chip excitation represents, to the best of our knowledge, one of the first empirical validations of compact data-driven thermal models at this scale.

6.5 Conclusion

This chapter presented a scalable empirical framework for power and thermal modeling of a modern many-core chiplet-based processor. The methodology operates without access to the vendor-disclosed core-level physical floorplan, while exploiting publicly available structural information.

A hierarchical power modeling strategy was first introduced to estimate per-core power traces from processor-level measurements. The resulting power model achieves an RMSE of 2.464 W under *PRBS* excitation, providing inputs for subsequent thermal identification.

A distributed MISO-ARX+noise identification methodology was then applied to a 56-core multi-chiplet processor. From structured excitation experiments, a global thermal coupling map was extracted and used to formulate floorplan reconstruction as a constrained quadratic assignment problem, enabling recovery of both chiplet partitioning and intra-chiplet core placement consistent with the imposed architectural constraints.

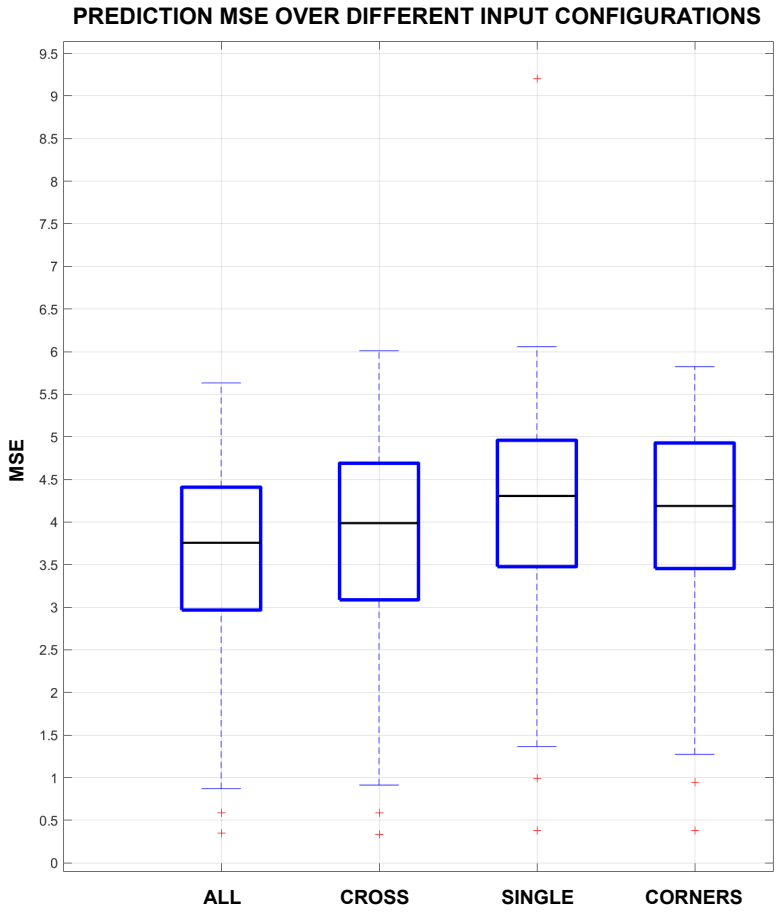


Figure 6.7: Distribution of MSE on prediction test for different input configurations of the thermal model.

Predictive validation under full-chip excitation provides an independent assessment of the inferred topology. The configuration exploiting all cores achieves a median MSE of $3.756\,^{\circ}\mathrm{C}^2$, while the single-core configuration yields $4.306\,^{\circ}\mathrm{C}^2$. Incorporating only Manhattan-1 neighbors reduces the median MSE to $3.988\,^{\circ}\mathrm{C}^2$, corresponding to a 7.4% improvement over the single-core case, whereas non-local chiplet corner cores provide only marginal gains.

Restricting the thermal model to Manhattan-1 neighbors drastically reduces the number of exogenous inputs—from 55 to at most 4 per core—while incurring only a limited increase in prediction error compared to the full model. This demonstrates that most of the relevant predictive information is spatially local and that substantial model simplification can be achieved without significant loss in accuracy.

Overall, the proposed framework demonstrates that compact, topology-aware thermal models can be derived directly from measurements on large-scale commercial chiplet-based processors, bridging the gap between small-scale monolithic studies and realistic many-core architectures.

Chapter 7

Context and Technological Framework

The research work presented in this thesis has been developed within the broader context of the European and international efforts toward open, energy-efficient, and HPC architectures.

The RISC-V ISA — released under an open license — represents a major technological shift in modern processor design. Its open nature allows unrestricted use, modification, and extension of the ISA, fostering a collaborative ecosystem across academia, industry, and open-source communities. Owing to its modularity and extensibility, RISC-V has rapidly gained traction not only in the embedded domain but also in HPC and data-center hardware, where heterogeneous and energy-efficient accelerators are increasingly required. The ability to design custom, domain-specific extensions while preserving software compatibility makes RISC-V a cornerstone technology for next-generation, power-aware computing systems [121].

Within this context, the **ControlPULP** platform, introduced in Chapter 3, has been jointly developed by the University of Bologna and ETH Zürich. ControlPULP is entirely based on the open RISC-V ISA and serves as a hardware–software PCS subsystem for fine-grained power and thermal management. It has been integrated within the **Rhea1 processor** developed by **SiPearl** as part of the **European Processor Initiative (EPI)** [27]. The Rhea1 processor, which successfully completed its tape-out in August 2025, represents the first European general-purpose HPC-class CPU designed to reach *exascale*-level performance while leveraging open and modular design principles. ControlPULP constitutes one of its key embedded controllers, enabling scalable and programmable management of power and thermal policies in compliance with RISC-V standards.

The second main contribution of this thesis, presented in Chapter 4, focuses on the implementation of the **SCMI** communication framework. This interface, developed and validated on Arm-based server platforms, provides a standardized

protocol for exchanging performance and power management directives between high-level operating system agents and low-level controllers. Interestingly, SCMI shares strong conceptual similarities with the emerging **RPMI** specification [100], making the developed framework a solid foundation for future RISC-V-based power management infrastructures.

Furthermore, both ControlPULP and the developed communication interfaces — including the **XRCE-based** introduced in chapter 5 for distributed power and thermal management coordination — will serve as core components within the upcoming **DARE project** (*Dependable and Secure Runtime for Distributed and Heterogeneous Computing*) [37]. Funded under the European Horizon framework, DARE aims to deliver an open and dependable software–hardware stack for next-generation heterogeneous HPC platforms, enabling adaptive and energy-aware runtime management across chiplets and accelerators. The reuse and extension of ControlPULP within DARE highlight the continuity and scalability of the research outcomes achieved in this thesis.

Chapter 8

Conclusions and Future Work

This dissertation has explored the design, implementation, and experimental characterization of scalable infrastructures for power and thermal management in modern many-core and chiplet-based processors. The proposed methodologies address the increasing complexity of hardware–software coordination in power and temperature-constrained computing systems, combining open hardware architectures, low-latency communication interfaces, and data-driven thermal–power models.

The first contribution introduced **ControlPULP**, an open and scalable RISC-V–based PCS designed to execute fine-grained control loops at sub-millisecond time scales. Its parallel microcontroller architecture and DMA-assisted sensing enable reproducible research on distributed, high-performance power and thermal control, marking a step toward open and programmable control planes for next-generation processors.

The second contribution focused on the quantitative characterization of standardized power-management interfaces. Through a FPGA-assisted HIL framework, this work provided the first detailed breakdown of latencies in SCMI-based delegation schemes, revealing the impact of firmware and protocol overheads on the responsiveness of dynamic voltage and frequency scaling. These results highlight the importance of low-latency control paths for stable and efficient runtime management.

The third contribution proposed a user-space communication framework for distributed PCS networks, based on the Micro XRCE-DDS middleware. By relocating the control path to user space and introducing a shared-memory transport, this approach removes kernel-mediated bottlenecks and enables flexible coordination among multiple controllers and application-level policy managers. This design demonstrates the feasibility of scalable, middleware-based control networks capable of spanning chiplets and subsystems within complex HPC architectures.

Finally, the fourth contribution developed a compact empirical methodology for power and thermal modeling in multi-chiplet processors. By leveraging distributed MISO-ARX identification, the proposed approach captures cross-chiplet thermal couplings directly from measurement data and reconstructs lightweight yet accurate models suitable for predictive control on commercial silicon such as Intel Sapphire Rapids. Together, these contributions provide a coherent framework that spans from hardware control substrates to communication interfaces and modeling methodologies, enabling open, reproducible exploration of energy management in heterogeneous computing systems.

Looking forward, several directions emerge from this research. A natural continuation involves extending the methodologies developed around SCMI toward the emerging RISC-V Platform Management Interface (RPMI), fostering open and standardized power-management frameworks for future RISC-V–based HPC processors. At the same time, the modeling techniques proposed here could be adapted to processors operating under advanced cooling solutions—such as liquid or hybrid systems—or to GPU-class accelerators characterized by high spatial thermal gradients and rapid transients. Building upon the Micro XRCE-DDS interface, further developments may also explore the integration of user-space power controllers with higher-level HPC runtime supervisors, enabling distributed policy enforcement based on telemetry aggregated from multiple subsystems. Finally, combining the empirical models developed in this work with predictive or learning-based strategies could lead to intelligent controllers capable of dynamically adapting to changing workloads, cooling conditions, and device aging.

In conclusion, this thesis advances the state of the art in power and thermal management by bridging open hardware, standardized communication, and data-driven modeling. Its results lay the groundwork for open, distributed, and adaptive management frameworks for future chiplet-based and heterogeneous high-performance computing systems.

Bibliography

[1] *AMBA AXI and ACE Protocol Specification Issue F.b*. ARM Ltd. URL: https://cv32e40p.readthedocs.io/en/latest/.

[2] AMD. *AMD EPYC™ 9004 and 8004 Series CPU Power Management*. White Paper. Accessed: 2025-10-24. Advanced Micro Devices, Inc., June 2024. URL: https://www.amd.com/content/dam/amd/en/documents/products/processors/server/epyc/epyc-8004-and-9004-series-cpu-power-management-white-paper.pdf.

[3] AMD. *EPYC 7003 Milan*. https://en.wikichip.org/wiki/amd/epyc. 2021.

[4] AMD. *EPYC 7004 Genoa*. https://en.wikichip.org/wiki/amd/cores/genoa.

[5] Arm. *Arm Corstone SSE-700 Subsystem*. 2020. URL: https://developer.arm.com/documentation/101418/0100/Message-Handling-Unit/Message-Handling-Unit-v2?lang=en.

[6] Arm. *Power Control System Architecture*. https://developer.arm.com/documentation/den0050/d/?lang=en. 2023.

[7] Arm. *SCP-firmware - version 2.13*. https://github.com/Arm-software/SCP-firmware. 2023.

[8] Arm Ltd. *AMBA APB Protocol Specification*. https://developer.arm.com/documentation/ihi0024. Version 2.0, Arm IHI 0024D. 2020.

[9] Arm Ltd. *AMBA AXI and ACE Protocol Specification*. https://developer.arm.com/documentation/ihi0022. Version 2.0, Arm IHI 0022E. 2023.

[10] Arm Ltd. *ARM Generic Interrupt Controller Architecture Specification, GICv2*. https://developer.arm.com/documentation/ihi0048/latest. Last accessed: 10/02/2026. 2023.

[11] Arm Ltd. *System Control and Management Interface (SCMI) Specification, Version 3.2*. https://developer.arm.com/documentation/. 2022.

[12] *Arm System Control and Management Interface v3.0*. ARM Ltd. URL: https://developer.arm.com/documentation/den0056/latest.

[13] D. Atienza et al. “A fast HW/SW FPGA-based thermal emulation framework for multi-processor system-on-chip”. In: *2006 43rd ACM/IEEE Design Automation Conference*. 2006, pp. 618–623. DOI: 10.1145/1146909.1147068.

[14] David Atienza. “Emulation-based transient thermal modeling of 2D/3D Systems-On-Chip with active cooling”. In: *2009 15th International Workshop on Thermal Investigations of ICs and Systems*. 2009, pp. 50–55.

[15] M Avgerinou, P Bertoldi, and L Castellazzi. *Trends in data Centre energy consumption under the European code of conduct for data Centre energy efficiency. Energies 10 (10): 1470*. 2017.

[16] Robert Balas, Alessandro Ottaviano, and Luca Benini. *CV32RT: Enabling Fast Interrupt and Context Switching for RISC-V Microcontrollers*. 2023. arXiv: 2311.08320 [cs.AR].

[17] Giovanni Bambini et al. “An Open-Source Scalable Thermal and Power Controller for HPC Processors”. In: *2020 IEEE 38th International Conference on Computer Design (ICCD)*. 2020, pp. 364–367.

[18] Giovanni Bambini et al. “Modeling the Thermal and Power Control Subsystem in HPC Processors”. In: *2022 IEEE Conference on Control Technology and Applications (CCTA)*. 2022, pp. 397–402. DOI: 10.1109/CCTA49430.2022.9966082.

[19] Richard Barry. “Mastering the FreeRTOS real time kernel”. In: *Real Time Engineers Ltd* (2016).

[20] A. Bartolini et al. “Thermal and Energy Management of High-Performance Multicores: Distributed and Self-Calibrating Model-Predictive Controller”. In: *IEEE Transactions on Parallel and Distributed Systems* 24.1 (2013), pp. 170–183.

[21] Andrea Bartolini et al. “A PULP-based Parallel Power Controller for Future Exascale Systems”. In: *2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*. 2019, pp. 771–774. DOI: 10.1109/ICECS46596.2019.8964699.

[22] Andrea Bartolini et al. “Monte Cimone: Paving the Road for the First Generation of RISC-V High-Performance Computers”. In: *arXiv preprint arXiv:2205.03725* (2022).

[23] Andrea Bartolini et al. “Thermal and energy management of high-performance multicores: Distributed and self-calibrating model-predictive controller”. In: *IEEE Transactions on Parallel and Distributed Systems* 24 (1 2013), pp. 170–183. ISSN: 10459219. DOI: 10.1109/TPDS.2012.117.

[24] F. Beneventi et al. “An Effective Gray-Box Identification Procedure for Multicore Thermal Modeling”. In: *IEEE Transactions on Computers* 63.5 (2014), pp. 1097–1110.

[25] Francesco Beneventi, Andrea Bartolini, and Luca Benini. "On-line thermal emulation: How to speed-up your thermal controller design". In: *2013 23rd International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS)*. 2013, pp. 99–106. DOI: 10.1109/PATMOS.2013.6662161.

[26] Francesco Beneventi et al. "An Effective Gray-Box Identification Procedure for Multicore Thermal Modeling". In: *IEEE Transactions on Computers* 63.5 (2014), pp. 1097–1110.

[27] Florian Berberich et al. *European HPC Landscape 2023*. Tech. rep. ETP4HPC, 2023.

[28] Christian Bienia et al. "The PARSEC Benchmark Suite: Characterization and Architectural Implications". In: *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*. PACT '08. Toronto, Ontario, Canada: Association for Computing Machinery, 2008, pp. 72–81. ISBN: 9781605582825. DOI: 10.1145/1454115.1454128. URL: https://doi.org/10.1145/1454115.1454128.

[29] Andrea Borghesi, Alessio Burrello, and Andrea Bartolini. "ExaMon-X: a Predictive Maintenance Framework for Automatic Monitoring in Industrial IoT Systems". In: *IEEE Internet of Things Journal* (2021), pp. 1–1. DOI: 10.1109/JIOT.2021.3125885.

[30] Nikolay Brayanov and Arno Eichberger. "Automation in Hardware-in-the-Loop Units Development and Integration". In: *2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*. 2019, pp. 191–197. DOI: 10.1109/QRS-C.2019.00047.

[31] Thomas Burd et al. ""Zeppelin": An SoC for Multichip Architectures". In: *IEEE Journal of Solid-State Circuits* 54.1 (2019), pp. 133–143. DOI: 10.1109/JSSC.2018.2873584.

[32] *burnP6(1) — Multicore CPU burn-in utility*. https://manpages.debian.org/jessie/cpuburn/burnP6.1.en.html. Accessed: 12-09-2025. Debian Jessie manpages, 2015.

[33] D. Cesarini et al. "COUNTDOWN: a Run-time Library for Performance-Neutral Energy Saving in MPI Applications". In: *IEEE Transactions on Computers* (2020), pp. 1–1.

[34] Daniele Cesarini et al. "Countdown slack: a run-time library to reduce energy footprint in large-scale MPI applications". In: *IEEE Transactions on Parallel and Distributed Systems* 31.11 (2020), pp. 2696–2709.

[35] Intel Corporation. *4th Gen Intel® Xeon® Scalable Processors (Codename Sapphire Rapids) Datasheet, Volume 1*. Tech. rep. Official Intel datasheet describing the architecture and multi-die design of Sapphire Rapids Xeon processors. Intel Corporation, Aug. 2024. URL: https://cdrdv2-public.intel.com/814093/814093_SPR_Datasheet_Vol1_Rev1p0.pdf.

[36] AM Coutinho Demetrios et al. “Performance and energy trade-offs for parallel applications on heterogeneous multi-processing systems”. In: *Energies* 13.9 (2020), p. 2409.

[37] *DARE Project: Dependable and Secure Runtime for Distributed and Heterogeneous Computing*. European Commission Horizon Europe Programme. https://cordis.europa.eu/project/id/101147234. 2025.

[38] Shidhartha Das, Paul Whatmough, and David Bull. “Modeling and characterization of the system-level Power Delivery Network for a dual-core ARM Cortex-A57 cluster in 28nm CMOS”. In: *2015 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. 2015, pp. 146–151. DOI: 10.1109/ISLPED.2015.7273505.

[39] R. H. Dennard et al. “Design of ion-implanted MOSFET’s with very small physical dimensions”. In: *IEEE Journal of Solid-State Circuits*. Vol. 9. 5. IEEE, Oct. 1974, pp. 256–268. DOI: 10.1109/JSSC.1974.1050511.

[40] Roberto Diversi, Andrea Bartolini, and Luca Benini. “Thermal Model Identification of Computing Nodes in High-Performance Computing Systems”. In: *IEEE Transactions on Industrial Electronics* 67.9 (2020), pp. 7778–7787.

[41] Roberto Diversi et al. “Bias-Compensated Least Squares Identification of Distributed Thermal Models for Many-Core Systems-on-Chip”. In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 61.9 (2014), pp. 2663–2676.

[42] ECS-Lab. *Examon Parser: MQTT Data Processing Engine*. https://gitlab.com/ecs-lab/sustainablehpc/examon_parser. Online repository, ECS-Lab Sustainable HPC Project. 2024.

[43] ECS-Lab. *PMU_PUB: Performance Monitoring Utility Publisher*. https://gitlab.com/ecs-lab/sustainablehpc/pmu_pub. Online repository, ECS-Lab Sustainable HPC Project. 2024.

[44] eProsima. *Micro XRCE-DDS: eXtremely Resource Constrained Environments DDS implementation*. https://github.com/eProsima/Micro-XRCE-DDS. Accessed Oct. 2025. 2024.

[45] eProsima S.L. *eProsima Micro-CDR: A lightweight C library implementing the CDR serialization standard for embedded systems*. Version 2.0.1. Accessed: 29 October 2025. 2024. URL: https://github.com/eProsima/Micro-CDR.

[46] Hadi Esmaeilzadeh et al. “Dark silicon and the end of multicore scaling”. In: *38th Annual International Symposium on Computer Architecture (ISCA)*. San Jose, CA, USA: IEEE, 2011, pp. 365–376. DOI: 10.1145/2000064.2000108.

[47] ExamonHPC. *Examon: Scalable Monitoring Framework for HPC Systems*. https://github.com/ExamonHPC/examon. Online repository, High-Performance Computing Monitoring Framework. 2024.

[48] The Linley Group. *SiPearl Develops ARM HPC Chip*. https://www.linleygroup.com/newsletters/newsletter_detail.php?num=6227&year=2020&tag=3. 2020.

[49] Andrew Grover. "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." In: *Queue* 1.7 (Oct. 2003), pp. 66–72. ISSN: 1542-7730. DOI: 10.1145/957717.957774. URL: https://doi.org/10.1145/957717.957774.

[50] Programmer Guide. *Intel 64 and IA-32 Architectures Software Developer's Manual*. 2022. URL: https://software.intel.com/content/www/us/en/develop/articles/intel-sdm.html.

[51] Steve Gunther et al. "Energy-Efficient Computing: Power Management System on the Nehalem Family of Processors". In: *Intel Technology Journal* 14.3 (2010), pp. 50–66.

[52] Gurobi Optimization, LLC. *Gurobi Optimizer Reference Manual*. Accessed: 2024-09-20. 2024. URL: https://www.gurobi.com.

[53] D. Hackenberg et al. "An Energy Efficiency Feature Survey of the Intel Haswell Processor". In: *2015 IEEE International Parallel and Distributed Processing Symposium Workshop*. 2015, pp. 896–904. DOI: 10.1109/IPDPSW.2015.70.

[54] John L. Hennessy and David A. Patterson. *Computer Architecture: A Quantitative Approach*. 5th ed. Amsterdam: Morgan Kaufmann, 2012. ISBN: 978-0-12-383872-8.

[55] IBM. *OpenPower OCC*. https://github.com/open-power/occ. 2022.

[56] Intel. *Power Management in Intel® Architecture Servers*. https://www.intel.com/content/dam/support/us/en/documents/motherboards/server/sb/power_management_of_intel_architecture_servers.pdf. 2009.

[57] Intel. *Raptor Lake*. https://en.wikichip.org/wiki/intel/microarchitectures/raptor_lake. 2022.

[58] Intel Corporation. *Intel Dynamic Platform and Thermal Framework (DPTF)*. https://github.com/intel/dptf. Accessed Oct. 2025. 2024.

[59] Intel Corporation. *thermald – Linux thermal management daemon*. https://github.com/intel/thermal_daemon. Accessed Oct. 2025. 2024.

[60] Hong Jiang. "Intel's ponte vecchio gpu: Architecture, systems & software". In: *2022 IEEE Hot Chips 34 Symposium (HCS)*. IEEE Computer Society. 2022, pp. 1–29.

[61] Norman P. Jouppi et al. "In-datacenter performance analysis of a Tensor Processing Unit". In: *44th Annual International Symposium on Computer Architecture (ISCA)*. Toronto, Canada: ACM/IEEE, 2017, pp. 1–12. DOI: 10.1145/3079856.3080246.

[62] Bartłomiej Kocot, Paweł Czarnul, and Jerzy Proficz. "Energy-aware scheduling for high-performance computing systems: A survey". In: *Energies* 16.2 (2023), p. 890.

[63] Andreas Kurth, Björn Forsberg, and Luca Benini. "HEROv2: Full-Stack Open-Source Research Platform for Heterogeneous Computing". In: *IEEE Trans. Parallel Distrib. Syst.* 33.12 (Dec. 2022), pp. 4368–4382. ISSN: 1045-9219. DOI: 10.1109/TPDS.2022.3189390. URL: https://doi.org/10.1109/TPDS.2022.3189390.

[64] Annapurna Labs. *AWS Graviton 2*. https://en.wikichip.org/wiki/annapurna_labs/alpine/alc12b00. 2020.

[65] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *Nature* 521.7553 (2015), pp. 436–444. DOI: 10.1038/nature14539.

[66] Charles E. Leiserson et al. "There's plenty of room at the Top: What will drive computer performance after Moore's law?" In: *Science* 368.6495 (2020). ISSN: 0036-8075. DOI: 10.1126/science.aam9744. eprint: https://science.sciencemag.org/content/368/6495/eaam9744.full.pdf. URL: https://science.sciencemag.org/content/368/6495/eaam9744.

[67] Linux Foundation. *CPUFreq subsystem in the Linux kernel*. https://www.kernel.org/doc/html/latest/admin-guide/pm/cpufreq.html. Accessed Oct. 2025. 2024.

[68] Yongpan Liu et al. "Accurate Temperature-Dependent Integrated Circuit Leakage Power Estimation Is Easy". In: *Proceedings of the Design, Automation & Test in Europe (DATE 2007)*. 2007, pp. 1–6.

[69] Yongpeng Liu and Hong Zhu. "A survey of the research on power management techniques for high-performance systems". In: *Software: Practice and Experience* 40 (2010).

[70] Zhifeng Liu and Hong Zhu. "A survey of the research on power management techniques for high-performance systems". In: *Softw., Pract. Exper.* 40 (Oct. 2010). DOI: 10.1002/spe.v40:11.

[71] Ampere Computing LLC. *Ampere Altra® Processors – Products*. https://amperecomputing.com/products/processors. Accessed: 2025-10-24. 2025.

[72] Google LLC. *Power management for multiple processor cores*. U.S. Patent US8402290B2, Dec. 2020.

[73] Arm Ltd. *Power and Performance Management using Arm SCMI Specification*. https://developer.arm.com/documentation/102886/001?lang=en. Aug. 2019.

[74] Runsheng Ma, Bei Yu, and David Z. Pan. "TAP: Thermal-Aware Placement for 2.5D Chiplet Design". In: *Proceedings of the 2021 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 2021, pp. 1–9.

[75] John McCalpin. “Mapping Core, CHA, and Memory Controller Numbers to Die Locations in Intel Xeon Phi x200 (” Knights Landing”,” KNL”) Processors”. In: (2021).

[76] John D. McCalpin. “Bandwidth Limits in the Intel Xeon Max (Sapphire Rapids with HBM) Processors”. In: *High Performance Computing*. Ed. by Amanda Bienz et al. Cham: Springer Nature Switzerland, 2023, pp. 403–413. ISBN: 978-3-031-40843-4.

[77] Wes McKinney et al. “Data structures for statistical computing in Python.” In: *scipy* 445.1 (2010), pp. 51–56.

[78] Fabio Montagna et al. “A Low-Power Transprecision Floating-Point Cluster for Efficient Near-Sensor Data Analytics”. In: *IEEE Transactions on Parallel and Distributed Systems* 33.5 (2022), pp. 1038–1053. DOI: 10.1109/TPDS.2021.3101764.

[79] Gordon E. Moore. “Cramming more components onto integrated circuits”. In: *Electronics* 38.8 (Apr. 1965). Reprinted in Proceedings of the IEEE, 1998, pp. 114–117. DOI: 10.1109/JPROC.1998.658762.

[80] Alexander A Moskovsky et al. “Server Level Liquid Cooling: Do Higher System Temperatures Improve Energy Efficiency?” In: *Supercomputing frontiers and innovations* 3.1 (2016), pp. 67–74.

[81] Matthias Müller et al. “SPEC Benchmarks”. In: *Encyclopedia of Parallel Computing*. Ed. by David Padua. Boston, MA: Springer US, 2011, pp. 1886–1893. ISBN: 978-0-387-09766-4. DOI: 10.1007/978-0-387-09766-4_370. URL: https://doi.org/10.1007/978-0-387-09766-4%5C_370.

[82] Almir Mutapcic et al. “Processor Speed Control with Thermal Constraints”. In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 56.9 (2009), pp. 1994–2008.

[83] Samuel Naffziger et al. “Pioneering Chiplet Technology and Design for the AMD EPYC™ and Ryzen™ Processor Families : Industrial Product”. In: *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. 2021, pp. 57–70. DOI: 10.1109/ISCA52012.2021.00014.

[84] OASIS. *MQTT Version 5.0*. Tech. rep. Message Queuing Telemetry Transport (MQTT) Protocol Specification. OASIS Standard, 2019. URL: https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html.

[85] Object Management Group. *Data Distribution Service (DDS) Specification*. https://www.omg.org/spec/DDS. Version 1.4. 2020.

[86] Herman Oprins, Vladimir Cherman, and E. Beyne. "Generic thermal modeling study of the impact of 3D-interposer material and thickness options on the thermal performance and die-to-die thermal coupling". In: *Fourteenth Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems (ITherm)*. 2014, pp. 72–78. DOI: 10.1109/ITHERM.2014.6892266.

[87] Alessandro Ottaviano et al. "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". In: *ArXiv* abs/2306.09501 (2023).

[88] Alessandro Ottaviano et al. "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". In: *International Journal of Parallel Programming* (Feb. 2024). ISSN: 1573-7640. DOI: 10.1007/s10766-024-00761-4. URL: https://doi.org/10.1007/s10766-024-00761-4.

[89] John D. Owens et al. "GPU computing". In: *Proceedings of the IEEE* 96.5 (May 2008), pp. 879–899. DOI: 10.1109/JPROC.2008.917757.

[90] David A. Patterson and John L. Hennessy. *Computer Organization and Design*. 2nd. Morgan Kaufmann Publishers, 1998, p. 715. ISBN: 15-586-0428-6.

[91] *Photolithography reticle-size limit and lithographic constraints*. https://www.semiwiki.com/. Accessed: 2025-09-30.

[92] Konstantin F. Pilz et al. "Trends in AI Supercomputers". In: *arXiv preprint arXiv:2504.16026* (2025).

[93] Federico Pittino et al. "A Scalable Framework for Online Power Modelling of High-Performance Computing Nodes in Production". In: *International Conference on High Performance Computing & Simulation (HPCS)*. IEEE. 2018, pp. 300–307.

[94] Federico Pittino et al. "Robust Identification of Thermal Models for In-Production High-Performance-Computing Clusters with Machine-Learning-Based Data Selection". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39.10 (2020), pp. 2042–2054.

[95] Jon Postel. *Transmission Control Protocol*. RFC 793. IETF, Sept. 1981. DOI: 10.17487/RFC0793. URL: https://www.rfc-editor.org/rfc/rfc793.

[96] Jon Postel. *User Datagram Protocol*. RFC 768. IETF, Aug. 1980. DOI: 10.17487/RFC0768. URL: https://www.rfc-editor.org/rfc/rfc768.

[97] Linux man-pages project. *proc_interrupts(5) — Linux manual page*. https://man7.org/linux/man-pages/man5/proc_interrupts.5.html. Accessed: 2026-02-04.

[98] Ismael Ripoll and Rafael Ballester. “Period Selection for Minimal Hyperperiod in Real-Time Systems”. In: 2014.

[99] RISC-V. *”Smclic” Core-Local Interrupt Controller (CLIC) RISC-V Privileged Architecture Extension*. URL: https://github.com/riscv/riscv-fast-interrupt/blob/master/clic.adoc.

[100] RISC-V International. *RISC-V Platform Management Interface (RPMI) Draft Specification*. https://github.com/riscv-non-isa/riscv-rpmi. 2024.

[101] RISC-V Non-ISA Working Group. *RISC-V Platform Management Interface Specification (RPMI)*. https://github.com/riscv-non-isa/riscv-rpmi. Accessed: 2025-10-04; OS-agnostic messaging interface for system management and control. 2025.

[102] Todd Rosedahl et al. “Power/Performance Controlling Techniques in OpenPOWER”. In: *High Performance Computing*. Ed. by Julian M. Kunkel et al. Cham: Springer International Publishing, 2017, pp. 275–289. ISBN: 978-3-319-67630-2.

[103] Davide Rossi et al. “PULP: A parallel ultra low power platform for next generation IoT applications”. In: *2015 IEEE Hot Chips 27 Symposium (HCS)*. 2015, pp. 1–39.

[104] Davide Rossi et al. “Ultra-Low-Latency Lightweight DMA for Tightly Coupled Multi-Core Clusters”. In: *Proceedings of the 11th ACM Conference on Computing Frontiers*. CF ’14. Cagliari, Italy: Association for Computing Machinery, 2014. ISBN: 9781450328708.

[105] Efraim Rotem et al. “Power-Management Architecture of the Intel Microarchitecture Code-Named Sandy Bridge”. In: *IEEE Micro* 32.2 (2012), pp. 20–27. DOI: 10.1109/MM.2012.12.

[106] Davide Schiavone. *OpenHW Group CV32E40P User Manual*. OpenHW Group. URL: https://cv32e40p.readthedocs.io/en/latest/.

[107] Martin Schlager, Roman Obermaisser, and Wilfried Elmenreich. “A Framework for Hardware-in-the-Loop Testing of an Integrated Architecture”. In: vol. 4761. May 2007, pp. 159–170. ISBN: 978-3-540-75663-7. DOI: 10.1007/978-3-540-75664-4_16.

[108] Robert Schöne et al. “Energy Efficiency Features of the Intel Skylake-SP Processor and Their Impact on Performance”. In: *2019 International Conference on High Performance Computing Simulation (HPCS)*. 2019, pp. 399–406. DOI: 10.1109/HPCS48598.2019.9188239.

[109] Vitor Ramos Gomes da Silva et al. “Analytical Energy Model Parametrized by Workload, Clock Frequency and Number of Active Cores for Share-Memory High-Performance Computing Applications”. In: *Energies* 15.3 (2022), p. 1213.

[110] Kevin Skadron et al. “Temperature-aware microarchitecture”. In: *30th Annual International Symposium on Computer Architecture (ISCA)*. San Diego, CA, USA: ACM/IEEE, 2003, pp. 2–13. DOI: 10.1145/859618.859621.

[111] Raja Swaminathan. “The next frontier: Enabling Moore’s Law using heterogeneous integration”. In: *Chip Scale Review* 26 (2022), pp. 1–8.

[112] System Management Interface Forum, Inc. *AVSBus Specification*. https://pmbus.org/specification-archives/. Last accessed: 10/02/2026. 2015.

[113] System Management Interface Forum, Inc. *PMBus Power Management Bus Specification*. https://pmbus.org/current-specifications/. Last accessed: 10/02/2026. 2015.

[114] Zhangxi Tan et al. “A Case for FAME: FPGA Architecture Model Execution”. In: *Proceedings of the 37th Annual International Symposium on Computer Architecture*. ISCA ’10. Saint-Malo, France: Association for Computing Machinery, 2010, pp. 290–301. ISBN: 9781450300537. DOI: 10.1145/1815961.1815999. URL: https://doi.org/10.1145/1815961.1815999.

[115] *The RISC-V Instruction Set Manual Volume II: Privileged Architecture*. RISC-V. URL: https://riscv.org/technical/specifications/.

[116] Andrea Tilli et al. “A two-layer distributed MPC approach to thermal control of Multiprocessor Systems-on-Chip”. In: *Control Engineering Practice* 122 (May 2022). ISSN: 09670661. DOI: 10.1016/j.conengprac.2022.105099.

[117] UEFI. *ACPI Specification 6.5*. https://uefi.org/specs/ACPI/6.5/. 2023.

[118] Ankush Varma et al. “Power management in the Intel Xeon E5 v3”. In: *2015 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. 2015, pp. 371–376. DOI: 10.1109/ISLPED.2015.7273542.

[119] Antonio del Vecchio et al. “Performance Characterization of Hardware/Software Communication Interfaces in End-to-End Power Management Solutions of High-Performance Computing Processors”. In: *Energies* 17.22 (2024). ISSN: 1996-1073. DOI: 10.3390/en17225778. URL: https://www.mdpi.com/1996-1073/17/22/5778.

[120] Qipan Wang et al. “ATPlace2.5D: Analytical Thermal-Aware Chiplet Placement Framework for Large-Scale 2.5D-IC”. In: *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*. ICCAD ’24. Newark Liberty International Airport Marriott, New York, NY, USA: Association for Computing Machinery, 2025. ISBN: 9798400710773. DOI: 10.1145/3676536.3676648. URL: https://doi.org/10.1145/3676536.3676648.

[121] Andrew Waterman and Krste Asanović. *The RISC-V Instruction Set Manual, Volume I–II*. RISC-V International. `https://riscv.org/technical/specifications/`. 2022.

[122] Xilinx. *Zynq UltraScale+ Device Technical Reference Manual*. Xilinx. URL: `https://www.xilinx.com/products/boards-and-kits/ek-u1-zcu102-g.html#resources`.

[123] Kai Zhu et al. "3D-ICE 4.0: Accurate and efficient thermal modeling for 2.5 D/3D heterogeneous chiplet systems". In: *arXiv preprint arXiv:2512.05823* (2025).

Funded by the European Union – Next Generation EU, Mission 4, Component 2, Investment 3.3 (Ministerial Decree 352/2022) CUP J33C22001440009