UNIVERSIDAD POLITÉCNICA DE MADRID
Escuela Técnica Superior de Ingenieros Industriales

Data-Driven Resource Management of Reconfigurable Multi-Accelerator Systems in the Cloud-Edge Continuum

TESIS DOCTORAL

Presentada para optar al título de Doctor por:

Juan Encinas Anchústegui

Máster Universitario en Electrónica Industrial
por la Universidad Politécnica de Madrid

Madrid, 2025

UNIVERSIDAD POLITÉCNICA DE MADRID
Escuela Técnica Superior de Ingenieros Industriales

Doctorado en Ingeniería Eléctrica y Electrónica

Data-Driven Resource Management of Reconfigurable Multi-Accelerator Systems in the Cloud-Edge Continuum

TESIS DOCTORAL

Presentada para optar al título de Doctor por:

Juan Encinas Anchústegui

Máster Universitario en Electrónica Industrial
por la Universidad Politécnica de Madrid

Bajo la dirección de:

Dr. José Andrés Otero Marnotes
Dr. Alfonso Rodríguez Medina

Madrid, 2025

Título: Data-Driven Resource Management of Reconfigurable Multi-Accelerator Systems in the Cloud-Edge Continuum

Autor: Juan Encinas Anchústegui

Programa de Doctorado: Ingeniería Eléctrica y Electrónica

Dirección de Tesis:

Dr. José Andrés Otero Marnotes

Dr. Alfonso Rodríguez Medina

Revisores Externos:

Tribunal de Tesis:

Fecha de Defensa de Tesis:

Agradecimientos

Todo se remonta a aquel día de septiembre en el que Alfonso, Andrés y yo quedamos en las mesas de “la piscina” para hablar de qué era eso del doctorado. De toda la conversación, lo que mejor recuerdo es la insistencia en que lo pensara bien: que era duro, sobre todo mentalmente, y que serían cinco años de enfrentarme, en muchos casos solo, a infinidad de nuevos retos.

Y es cierto que una tesis es, en gran medida, un trabajo individual; pero, por suerte, durante estos cinco años he tenido a mucha gente detrás, apoyándome, animándome y hasta sufriendo conmigo en el camino. Esta sección va dedicada a todos ellos. Sois muchos, así que seré un poco general, pero vosotros ya sabéis quiénes sois.

Empiezo, como no podía ser de otra forma, por mis tutores, Alfonso y Andrés. Cinco añitos... vaya viaje; pero ya os decía yo que se confiaba. La verdad es que no me imagino unos jefes mejores. Y digo “jefes” porque así es sobre el papel, aunque sabéis que os considero más bien amigos. Voy a echar de menos esas reuniones de una hora en las que la primera mitad es una competición por ver quién dice más tonterías. Ahora, en serio: gracias por darme la oportunidad, por remar conmigo hasta el último minuto y, sobre todo, por confiar siempre en mí, incluso cuando yo mismo no lo hacía. No sé si nuestros caminos académicos se separarán en algún momento, pero ya sabéis que me tenéis para lo que sea.

Quiero hacer también una mención especial a Edu, el germen de todo esto. Empecé el máster un poco de casualidad y sin tener muy claro si era para mí. Por suerte para todos, ahí estaba Edu (literalmente en todas las asignaturas), haciendo que hasta el tema más aburrido del mundo pareciera de lo más interesante. Sinceramente, creo que fue esa pasión por la investigación que siempre transmites lo que se me fue pegando poco a poco, hasta que ya no hubo vuelta atrás. Es una pena que los nuevos estudiantes se lo vayan a perder, no entenderán nunca la suerte que tuvimos algunos...

Tras cinco años de tesis, la pregunta que más me hace la gente es si voy a acabar algún día, si no estoy cansado, si no quiero hacer otra cosa. La realidad es que, aunque ha sido mucho tiempo, a mí se me ha pasado volando. Y el motivo, una razón fundamental para que haya terminado esta tesis, es la gente que viene día a día al CEI: profesorado, personal de administración y compañeros de trabajo.

Creo que esa es la magia del CEI, y la razón por la que cada día crece más: aunque haya que publicar más y sacar más proyectos, siempre hay alguien en quien apoyarte (aunque sea porque te han dejado encerrado en la escuela). Al final, todos los problemas pesan menos si sabes que esa tarde toca chirin.

Aunque no puedo mencionar a todos, tengo que dar las gracias especialmente a Iñigo: he perdido la cuenta de las veces que has dedicado tu tiempo para que esta tesis saliera adelante.

On a more international note, I'd like to express my gratitude to each member of Professor Francesca Palumbo's group, who received me in Cagliari during my international stay and welcomed me as one of their own.

Por supuesto, no me olvido de toda esa gente que ha tenido que estar estos cinco años compartiéndome con la tesis.

Muchas gracias a mis amigos, que han tenido que sufrir innumerables veces que cancelara planes o llegara varias horas tarde porque "tenía que trabajar en la maldita tesis", y que, aun así, han seguido ahí, animándome y apoyándome en todo momento.

Y cierro con mi familia. Gracias por haber estado ahí desde el principio, por apoyarme incondicionalmente en cada paso, aunque a veces ni supiera explicaros bien qué estaba haciendo. Esta tesis es tan mía como vuestra. Os quiero mucho.

Resumen en español

Esta Tesis aborda la integración y gestión de recursos reconfigurables en el continuo *cloud-edge*. Se enfoca en sistemas reconfigurables multiacelerador sobre *Field-Programable Gate Arrays* (FPGAs), donde la *reconfiguración dinámica parcial* (DPR) permite explotar el paralelismo a nivel de datos (varias réplicas) y de tareas (varias tareas). La Tesis introduce una infraestructura que despliega y monitoriza cargas de trabajo dinámicas en nodos FPGA heterogéneos, desde placas de gama baja hasta dispositivos destinados al *cloud*, sin ajustes entre plataformas. La infraestructura amplía el *framework* ARTICo^3, soportando dispositivos para el *cloud* y ejecución multiusuario; emplea un modelo cliente-servidor que coordina la aceleración en FPGA entre múltiples usuarios; y empaqueta aceleradores y software en contenedores orquestados con Kubernetes y Liqo para gestionar el movimiento y escalado de tareas de forma transparente, bajo restricciones de latencia, rendimiento o consumo. Se incluye un *framework* de monitorización que genera trazas de consumo y rendimiento, sin introducir penalizaciones de rendimiento en el sistema.

Para afrontar la interacción entre *kernels* que se ejecutan en paralelo, esta Tesis propone una metodología de caracterización en tiempo de ejecución que entrena modelos basados en datos, para predecir consumo y rendimiento bajo los efectos de la interacción entre *kernels*. Se emplean modelos de *aprendizaje automático* (ML) incrementales que se actualizan durante la ejecución del sistema, evitando reentrenamientos completos cuando cambian las condiciones del sistema. La gestión del entrenamiento de estos modelos se realiza con un mecanismo de orquestación dedicado, que limita el impacto de los modelos en el sistema, reduciendo el impacto del modelado del >20% en alternativas de aprendizaje continuo a <5%, manteniendo la precisión de predicción dentro del 4% respecto al enfoque continuo. Los modelos son agnósticos al dispositivo y se han validado con distintas arquitecturas, métodos de reconfiguración y de medida de consumo, sin requerir ajustes por dispositivo.

Basada en estas predicciones, una metodología adaptativa gestiona la planificación de tareas sobre los recursos de la FPGA, con un enfoque multiobjetivo. Se emplea la metaheurística *Crow Search Algorithm*, adaptada a la naturaleza discreta del problema de planificación en FPGA, y explora el espacio de soluciones, jugando entre rendimiento, energía y reparto justo de recursos. Las soluciones candidatas se evalúan con los modelos en tiempo de ejecución, que consideran el comportamiento aislado y el impacto de la interferencia entre *kernels* para tomar decisiones. En el peor escenario, esta optimización reduce el tiempo de ejecución en un 11% frente a implementaciones no adaptativas, logrando ahorros energéticos proporcionales, y tiempos de espera significativamente menores cuando se prioriza el reparto justo de los recursos.

Al combinar virtualización, monitorización, modelado y planificación de tareas, esta Tesis aporta una implementación *open-source* de extremo a extremo que abstrae los detalles del hardware y es escalable en el continuo *cloud-edge*. La virtualización basada en contenedores habilita el uso de recursos remotos de forma transparente. La capa de monitorización proporciona las trazas necesarias para comprender el comportamiento del sistema. La capa de caracterización transforma las trazas en predicciones precisas de consumo y rendimiento. Por último, el planificador utiliza esas predicciones para guiar decisiones en tiempo de ejecución que minimizan las penalizaciones por interacción entre kernels. Una validación extensa (basada en *benchmarks* y casos de uso integrados) muestra que esta metodología mantiene sus beneficios en diferentes dispositivos, soporta escenarios tanto de alto rendimiento como de bajos recursos y ofrece una base sólida para la implementación de aceleración en FPGA en entornos modernos del continuo *cloud-edge*.

Abstract

This Thesis targets the practical integration and resource management of reconfigurable computing into the cloud-edge continuum. The focus is on reconfigurable multi-accelerator systems on Field-Programmable Gate Arrays (FPGAs), where the use of Dynamic and Partial Reconfiguration (DPR) enables the exploitation of data-level parallelism (i.e., parallel replicas) and task-level parallelism (i.e., parallel tasks). This Thesis introduces a platform-agnostic infrastructure that deploys and monitors dynamic workloads on heterogeneous FPGA nodes, from low-end embedded boards to cloud-grade cards, without platform-specific changes. The infrastructure extends the ARTICo^3 framework for cloud support and multi-tenant operation; uses a client/daemon execution model to coordinate FPGA acceleration between multiple users while enforcing isolation; and packages accelerators and software into containers orchestrated with Kubernetes and Liqo for seamless multi-cluster resource sharing, migration and scaling under latency, throughput, or power constraints. A modular monitoring framework exposes synchronized power and performance traces with high-resolution and lightweight modes, keeping low overhead across platforms.

To cope with interference among concurrently running kernels, the Thesis proposes a run-time workload-characterization methodology that learns data-driven models of power and performance under kernel-interaction effects. Starting from offline feasibility studies, it advances to incremental Machine Learning (ML) models that update online as the system operates, avoiding full retraining when workloads or platform conditions change. A dedicated learning-orchestration mechanism handles model updates to avoid competing with accelerator execution, reducing modeling overhead from >20% in continuous learning alternatives to <5% while keeping prediction accuracy within 4% of the continuous approach. The models are device-agnostic and have been validated on boards with diverse architectures and power-measurement capabilities, requiring minor device-specific tuning.

Built on these predictions, an adaptive, conflict-aware workload optimization methodology addresses task scheduling as a multi-objective optimization over discrete FPGA resources. The strategy employs the Crow Search Algorithm (CSA) metaheuristic, adapted to the discrete nature of the FPGA scheduling problem, and explores the solution space, trading between makespan, energy and fairness. Candidate solutions are evaluated with the run-time models, which account for standalone behavior and interference-induced slowdowns to make decisions. A comprehensive sensitivity analysis guides parameter settings to meet different operational goals. Even under worst-case conditions, the scheduler reduces total execution time by up to 11% relative to non-adaptive baselines, yielding proportional energy savings that are essential on resource-constrained edge devices, and significantly lower waiting times when fair use of resources is prioritized.

By combining virtualization, monitoring, modeling and scheduling, this Thesis delivers an open-source, end-to-end approach that abstracts hardware details and scales across the cloud-edge continuum. Container-based virtualization enables seamless migration and transparent use of remote resources. The monitoring layer provides the run-time traces needed to understand system behavior. The characterization layer turns traces into accurate predictions of power and performance. Finally, the scheduler uses those predictions to guide run-time decisions that minimize interaction penalties. Extensive validation, from dwarf-inspired High-Level Synthesis (HLS) benchmarks to integrated use cases, shows that the methodology sustains its benefits without per-platform tuning, supports both high-performance and resource-constrained scenarios and offers a foundation to practical, portable FPGA acceleration in modern cloud-edge continuum environments.

Extended Abstract

Over the past decades, computing has evolved from simple systems capable of executing one instruction at a time to highly parallel architectures designed to meet the growing demand for performance. In the early years, improvements were driven by increasing clock frequencies, supported by advances in semiconductor manufacturing that followed Moore's Law and Dennard Scaling. These trends enabled processors to become faster with each generation, but as transistor sizes approached atomic scales, physical and power limitations prevented further frequency scaling. At that point, the industry shifted its focus toward parallel computing, exploiting simultaneous execution to achieve higher performance. This triggered the rise of a variety of platforms, ranging from general-purpose multi-core processors and massively parallel Graphics Processing Units (GPUs) to specialized hardware such as Application-Specific Integrated Circuits (ASICs) and Field-Programmable Gate Arrays (FPGAs).

FPGAs combine the performance and efficiency of custom hardware with the flexibility of software. They can exploit both task-level and data-level parallelism and can be reconfigured at run time. This is known as Dynamic and Partial Reconfiguration (DPR), and it enables a single device to host multiple accelerators over time, replicate them to increase throughput, or change its configuration to suit the needs of a time-varying workload. In the context of this Thesis, a computing system with such capabilities is referred to as a reconfigurable multi-accelerator system. Such flexibility enables trade-offs between power, performance and resource usage, making FPGAs suitable for a wide range of applications, from low-latency real-time systems to low-power embedded devices.

While hardware capabilities have advanced, computing infrastructures have also undergone significant transformation in recent years. The emergence of cloud computing in the early 2000s shifted processing power from local machines to extensive centralized facilities. This model provided virtually unlimited scalability and access to high-performance resources without the need for dedicated on-site hardware. However, not every workload is suitable for execution in the cloud. Applications that require real-time responses may suffer from latency when data must travel to and from distant data centers. Others, such as those processing sensitive information, may be restricted by privacy or security concerns. These limitations motivated the development of the edge computing paradigm, where processing is performed closer to data sources. Edge computing reduces latency and can improve privacy, but it also introduces strict constraints on size, cost and power consumption, which in turn motivates the need for more advanced workload distribution and device operation optimization techniques.

To bridge the gap between centralized and distributed computing, the concept of the cloud-edge continuum has emerged. In this scenario, computing resources are

viewed as a unified entity spanning from the smallest edge nodes to the most powerful cloud servers. Workloads can be dynamically assigned to different points along this continuum depending on their requirements and the state of the system. Latency-sensitive or privacy-critical tasks may run at the edge, while more computationally intensive tasks can be offloaded to the cloud. Many FPGA vendors now provide devices tailored for both cloud and edge deployments, leveraging their performance, flexibility and efficiency benefits in such environments.

Integrating FPGAs into this continuum, however, presents two major challenges. First, workloads must be able to run on any FPGA node regardless of the specific device or architecture, which requires virtualization support capable of hiding hardware details while managing low-level operations such as DPR and Direct Memory Access (DMA) transfers. Second, DPR must be supported so that a single FPGA can be shared among multiple applications and users. In addition, workloads in this cloud-edge continuum environment are inherently dynamic: tasks can arrive at any time, resource availability changes frequently, and scheduling decisions must be made at run time. Therefore, static scheduling or design-time optimization is not enough. FPGAs need the ability to deploy workloads transparently, monitor their performance and power consumption in run time, learn from this data, and adapt task scheduling accordingly.

This Thesis proposes an infrastructure that addresses these challenges by supporting the deployment, execution, and management of dynamic workloads on reconfigurable multi-accelerator systems in a manner that is agnostic to the underlying hardware. It extends the ARTICo^3 framework for dynamically reconfigurable systems with new capabilities that make it suitable for both cloud-grade FPGAs and multi-tenant execution scenarios, ensuring that the same application can run on devices ranging from low-end embedded boards to high-performance datacenter cards without modification. To handle the particularities of FPGA-based systems for the cloud-edge continuum, the infrastructure integrates a client/daemon execution model that coordinates hardware-specific procedures, such as DPR and DMA transfers, while enforcing isolation between tenants. Moreover, workloads are encapsulated into containerized environments and orchestrated via Kubernetes, with Liqo, an open-source Kubernetes multi-cluster solution, enabling the seamless sharing of resources across multiple clusters. This setup allows applications to be transparently moved between geographically distributed nodes according to real-time constraints such as latency, throughput, or power budgets. Alongside this, a run-time modular monitoring framework is embedded directly into the infrastructure. This framework provides both high-resolution and lightweight monitoring options, enabling the accurate collection of synchronized performance and power traces while adapting to the capabilities and constraints of each platform.

The result is an infrastructure that not only abstracts the complexity of hardware acceleration deployment but also scales efficiently across heterogeneous platforms. Validation campaigns demonstrate minimal overhead when deploying hardware-accelerated tasks across heterogeneous devices, and a wide range of trade-offs between performance, energy consumption, and resource usage, making it a sound

foundation for integrating reconfigurable devices into the cloud-edge continuum.

Moreover, this Thesis introduces a workload characterization methodology for modeling the main metrics related to the execution of dynamic workloads in reconfigurable multi-accelerator systems at run time. In multi-tenant scenarios, where several applications may be deployed in parallel on the same device (e.g., FPGAs, but the same problem can also be identified in multi-core processors), multiple hardware kernels may share memory bandwidth, interconnects and Central Processing Units (CPUs). Interference between concurrently executing accelerators can have a substantial impact on performance and energy consumption. These kernel interaction effects are often unpredictable (at design time) and can lead to significant execution time variations (up to 5× variations have been observed during the experiments of this Thesis, compared to an isolated execution). To anticipate these effects, a data-driven characterization methodology is introduced in this Thesis.

The approach begins with an offline modeling strategy used to assess the feasibility of predicting kernel interactions from run-time monitoring traces. Building on these results, the methodology is extended into a run-time characterization process based on incremental Machine Learning (ML) models. These models are continuously updated during system operation to adapt to changes in workloads or hardware conditions, without the cost of retraining from scratch. To limit the overhead of incremental learning, a dedicated orchestration mechanism manages model updates in a way that avoids interfering with accelerator execution. This reduces the modeling overhead from over 20% in continuous learning approaches to less than 5%, while maintaining prediction accuracy within 4% of the continuous learning approach. In the experiments, this method achieves a Mean Absolute Percentage Error (MAPE) error below 0.6% when predicting the impact of complex kernel interaction patterns. Additionally, the methodology has been validated on devices with diverse architectures, reconfiguration methods, and power measurement capabilities, ensuring accuracy and low overhead without requiring platform-specific tuning. This portability ensures that the models can be deployed across the heterogeneous FPGA landscape of the cloud-edge continuum, providing run-time insight into workload behavior under dynamic operating conditions.

Building upon these characterization capabilities, this Thesis implements an adaptive scheduling strategy for reconfigurable multi-accelerator systems. In the dynamic and heterogeneous environment of the cloud-edge continuum, workloads cannot be predetermined at design time, and resource availability changes constantly when dealing with dynamic workloads and multi-tenant execution. Efficient execution in such conditions requires scheduling decisions that can adapt at run time, continuously optimizing performance and energy efficiency.

The proposed scheduling approach addresses the adaptive scheduling problem as a multi-objective optimization task, where the goal is to determine how and when tasks should be mapped across available reconfigurable resources to meet the user's needs. To solve this problem, a conflict-aware strategy is developed using the Crow Search Algorithm (CSA), a population-based metaheuristic adapted to the discrete nature of

the FPGA scheduling problem. Candidate solutions, representing possible task-to-resource assignments, are evaluated using the predictive models produced by the run-time characterization layer, which accounts for both individual kernel performance and kernel interaction effects.

A detailed sensitivity analysis is performed to explore how CSA parameters influence convergence speed and solution quality. These experiments guide the tuning of the scheduler to achieve different operational goals, including minimizing makespan, improving energy efficiency, and reducing waiting time. Results show that, even under a worst-case scenario, the scheduler reduces total execution time by up to 11% compared to non-adaptive approaches, yielding proportional energy savings in resource-constrained edge devices. When configured to prioritize fair use of resources, the strategy significantly reduces task waiting times, improving predictability for shared FPGA resources.

By integrating these components, this Thesis enables FPGA workload management in the cloud-edge continuum. Container virtualization abstracts the hardware differences, enabling seamless workload migration and scaling. The monitoring framework provides the data needed to understand system behavior, the characterization layer translates this data into accurate predictions of performance and power, and the scheduler uses those predictions to guide adaptive and optimal resource allocation.

Extensive validation across a range of devices (from low-end boards to high-end cloud devices) demonstrates that the proposed methodology maintains its performance and energy benefits without requiring platform-specific tuning. It scales across heterogeneous clusters, adapts to changing workloads, and supports both high-performance and resource-constrained deployments. Beyond the implementation, the methodology provides a reproducible approach that facilitates the practical deployment of reconfigurable computing in modern distributed environments.

The main contributions of this Thesis are summarized below:

- **[Infrastructure]** An open-source platform-agnostic infrastructure to deploy and monitor dynamic workloads on reconfigurable multi-accelerator systems in the cloud-edge continuum.
- **[Infrastructure]** The extension of the ARTICo^3 framework for cloud-based FPGAs and multi-tenant execution.
- **[Infrastructure]** A virtualization methodology to support the seamless deployment of hardware accelerators across the cloud-edge continuum.
- **[Monitoring]** An open-source, composable monitoring framework for synchronized power consumption and performance traces acquisition on reconfigurable multi-accelerator systems.
- **[Kernel Interaction Analysis]** An evaluation of the impact on power consumption and performance of the interaction between kernels when executing dynamic workloads in reconfigurable multi-accelerator systems.

- **[Modeling]** A device-agnostic, data-driven methodology to predict power consumption and performance in reconfigurable multi-accelerator systems.
- **[Modeling]** An incremental data-driven modeling extension to predict power consumption and computing performance in reconfigurable multi-accelerator systems at run time and on demand.
- **[Modeling]** A resource-aware model orchestration mechanism to perform run-time adaptive incremental learning and keep models updated while minimizing system overhead.
- **[Scheduling]** A metaheuristic-based strategy that relies on data-driven predictive models to schedule dynamic workloads in reconfigurable multi-accelerator systems, optimizing and trading off energy and performance on the one hand, and fair use of resources on the other hand, at run time.
- **[Scheduling]** A scheduling policy that uses run-time kernel interaction estimations to guide scheduling decisions in reconfigurable multi-accelerator systems.
- **[Cloud-Edge Continuum]** A methodology to grant user-agnostic compatibility across the diverse set of hardware nodes present in the cloud-edge continuum.
- **[Cloud-Edge Continuum]** A multi-cluster cloud-edge continuum architecture that integrates workload management, characterization and optimization, enabling the exploration of a vast amount of operating points and trade-offs.
- **[Validation]** A dwarf-based characterization and validation strategy for the proposed technology based on High-Level Synthesis (HLS) benchmarks.

This Thesis is structured into six chapters. Chapter 1 introduces the motivation, objectives, and research context of the work, along with the technological background on reconfigurable computing, cloud-edge continuum and scheduling techniques. Chapter 2 presents the workload management infrastructure designed for heterogeneous reconfigurable systems, describing the extensions to the ARTICo^3 framework to support cloud-based FPGAs and multi-tenant execution. Then, container-based workload deployment with Kubernetes and multi-cluster management through Liqo is presented, along with an integrated run-time monitoring framework. Chapter 3 details the run-time workload characterization methodology, which models the impact of kernel interactions through an incremental ML-based approach with a dedicated orchestration mechanism. Chapter 4 describes the adaptive conflict-aware scheduling strategy based on the CSA, which leverages run-time characterization to perform multi-objective workload optimization, supported by a sensitivity analysis of its parameters and validated on platforms ranging from edge to cloud. Chapter 5 addresses the integration of these components, conducting an experimental evaluation of the integrated architecture through two dedicated use cases. Finally, Chapter 6 summarizes the conclusions, highlights the main contributions, discusses their impact, and outlines potential future research directions.

Contents

List of Figures

List of Tables

Acronyms

ACO	Ant Colony Optimization
ADC	Analog-to-Digital Converter
AES	Advance Encryption Standar
AI	Artificial Intelligence
ALU	Arithmetic Logic Unit
API	Application Programming Interface
ARTICo^3	Arquitectura Reconfigurable para el Tratamiento Inteligente de Cómputo, Consumo y Confiabilidad
ASIC	Application-Specific Integrated Circuit
AWS	Amazon Web Service
AXI	Advance eXtensible Interface
BRAM	Block RAM
BMM	Block Matrix Multiplication
CB	Connection Box
CLB	Configurable Logic Block
CGRA	Coarse-Grained Reconfigurable Array
CNN	Convolutional Neural Network
CMS	Card Management Solution
CPS	Cyber Physical System
CPU	Central Processing Unit
CSA	Crow Search Algorithm
DAG	Directed Acyclic Graph

DMA Direct Memory Access

DMR Dual Modular Redundancy

DNN Deep Neural Network

DSE Design Space Exploration

DSP Digital Signal Processor

DPE Design and Programming Environment

DPR Dynamic and Partial Reconfiguration

DVFS Dynamic Voltage and Frequency Scaling

FCFS First Come, First Served

FF Flip-Flop

FFT Fast Fourier Transform

FPGA Field-Programmable Gate Array

FPS Frames per Second

FPW FPS per Watt

FSM Finite State Machine

FU Functional Unit

FUSE Front-end USEr

GA Genetic Algorithm

GPU Graphics Processing Unit

HBICAP High Bandwidth ICAP

HDL Hardware Description Language

HLS High-Level Synthesis

HPC High-Performance Computing

I²C Inter-Integrated Circuit

ICAP Internal Configuration Access Port

ILA Integrated Logic Analyzer

IoT Internet of Things

IP Intellectual Property

ISA Instruction Set Architecture

KNN K-Nearest Neighbor

LR Linear Regression

LSTM Long Short-Term Memory

LUT Look-Up Table

MAPE Mean Absolute Percentage Error

ML Machine Learning

MPSoC Multiprocessor System-on-Chip

MYRTUS Multi-layer 360° dYnamic orchestration and interopeRable design environmenT for compute-continUum Systems

NN Neural Network

NMP Near-Memory Processing

OCI Open Container Iniciative

OpenCL Open Computing Language

OS Operating System

PaaS Platform-as-a-Service

PCA Principal Component Analysis

PCAP Processor Configuration Access Port

PCIe Peripheral Component Interconnect Express

PE Processing Element

PL Programmable Logic

PMBus Power Management Bus

PMC Performance Monitoring Counter

POSIX Portable Operating System Interface

PS Processing System

PSO Particle Swarm Optimization

QoS Quality of Service

RF Random Forest

RL Reinforcement Learning

RLS Recursive Least Squares

RMSE Root Mean Squared Error

RR Round-Robin

RSB Reconfigurable Streaming Block

RT Regression Tree

RTE Regression Tree Ensemble

RTL Register-Transfer Level

S-NUCA Static Non-Uniform Cache Architecture

SB Switch Box

SIMD Single Instruction, Multiple Data

SIMT Single Instruction, Multiple Threads

SoM System on Module

SoC System on Chip

SoPC System on Programmable Chip

SPI Serial Peripheral Interface

SRAM Static RAM

SVM Support Vector Machine

SVR Support Vector Regression

TCL Tool Command Language

TMR Triple Modular Redundancy

UAV Unmanned Aerial Vehicle

USB Universal Serial Bus

VAPRES Virtual Architecture for Partially Reconfigurable Embedded Systems

XRT Xilinx RunTime

Chapter

1 INTRODUCTION

This introductory chapter provides the context surrounding the challenges addressed in this Thesis. It starts with a description of the motivation and goals driving the realization of this work, followed by a brief introduction to the technologies relevant to this Thesis. It concludes with an overview of the technical development undertaken, the document structure, and a summary of the content provided in each chapter.

1.1 Motivation

In the early days of computing, processors handled tasks sequentially by executing one instruction at a time. For example, a microcontroller in a washing machine would manage the rotation of the motor, read input sensors, and regulate the water temperature, executing these tasks in a fixed sequence. Since user applications at that time were reasonably straightforward, that approach was sufficient.

Over the following decades, processors were applied to a broader range of applications such as home computers, automatic automobile controls, or portable communication equipment, raising the need for higher computing power [Moore'64]. In response, the semiconductor manufacturing process underwent significant evolution. These advancements increased the number of transistors present in each new generation of processors, leading to the formulation of the empirical Moore's Law [Moore'06], which states that transistor density nearly doubles every 18 months in electronic devices. Transistor size, in turn, was consistently reduced, motivating the Dennard Scaling rule [Dennard'74], another empirical law stating power density remains constant in electronic devices as transistor size decreases. Combining both laws allowed the semiconductor industry to increase clock frequencies generation after generation, effectively enhancing computing performance and adapting to new application requirements.

However, when transistor size reached atomic levels, Dennard Scaling was no longer held, mainly due to quantum effects and other physical phenomena that prevented the expected proportional reduction in voltage and power [Bohr'07]. Moore's Law, in contrast, remains valid today, although some voices in the industry claim it will not hold much longer. Since clock frequencies cannot be increased

while keeping power consumption affordable, the semiconductor industry found the solution in a new computing paradigm: parallel computing. From then on, performance improvements have shifted from achieving faster sequential operations to exploiting parallel execution [Parkhurst'06, Akhter'06].

In this era of parallel computing, various computing devices have appeared to exploit different levels of parallelism. Software-based solutions such as multi-core processors or many-core Graphics Processing Units (GPUs) are characterized by their flexibility and programmer-friendly nature. Hardware-based solutions, such as Field-Programmable Gate Arrays (FPGAs) and domain-specific architectures like Application-Specific Integrated Circuits (ASICs), designed for a specific purpose, provide better performance than their software-based counterparts [Hennessy'17].

The hardware-like performance obtained from FPGAs renders a significantly better performance-to-power ratio compared to software-centric alternatives, which, together with the software-like flexibility enabled by their programmable logic, makes them the most suitable option for many application domains (e.g., real-time applications with low-latency requirements or edge applications with low-power constraints) [Nurvitadhi'16, Qasaimeh'19, Boutros'20]. Moreover, many current FPGAs offer a technique known as Dynamic and Partial Reconfiguration (DPR), which allows a portion of the programmable logic to be reconfigured at run time while the rest remains operational. This feature increases flexibility and exploits time multiplexing benefits, promoting smaller and cheaper FPGAs as resources' footprint is reduced (i.e., accelerator logic can be configured only when needed), and enabling latency/throughput trade-offs by dynamically adjusting the level of parallelism (i.e., multiple parallel replicas of the same accelerator to increase throughput or multiple different accelerators in parallel to reduce their latency) [Koch'12].

At the same time, computing infrastructures have been continuously evolving since their inception. In the early 2000s, cloud computing emerged as a dominant model that replaced locally-hosted computing infrastructures, centralizing processing power in large-scale computing facilities. Due to their virtually unlimited scalability, cloud platforms enabled access to high-performance computing resources without needing dedicated on-site hardware [Grossman'09]. It soon became apparent, however, that not every workload was well-suited for the cloud [Sadiku'14]. Latency became a significant constraint, as many applications, such as autonomous systems or healthcare devices, required real-time processing, where delays caused by transmitting data to and from the cloud were unacceptable. Privacy and security further limited the cloud, particularly for applications handling sensitive data (e.g., healthcare or financial applications). These challenges promoted the edge computing paradigm, where computing resources were placed closer to where data is produced, reducing latency, increasing security and privacy, and avoiding reliance on cloud connectivity [Satyanarayanan'17]. However, this new paradigm also presents challenges. For example, in Internet of Things (IoT) networks with a multitude of nodes, cost is a major concern. In mobile robotics, in turn, size and weight constraints are fundamental. Power consumption is a general restriction at the edge. All of these

constraints significantly limit the computing capabilities of edge devices compared to the cloud. Therefore, workloads must be balanced between cloud and edge environments based on performance, latency, and energy constraints [Mittal'17].

In the late 2010s, the concept of the cloud–edge continuum was introduced to bridge the gap between centralized and distributed computing [Javed'18], enabling the dynamic allocation of resources across the cloud, the edge, and the intermediate layers [Khalyeyev'23]. Instead of treating cloud and edge as distinct entities, the entire infrastructure is viewed as a continuum of computing resources, where workloads are distributed and deployed based on their real-time needs. Computationally intensive tasks are moved towards the cloud, while latency-, privacy-, and power-sensitive tasks are placed closer to the edge. Throughout this computing infrastructure evolution, many FPGAs vendors, motivated by the flexibility, performance, and energy efficiency advantages provided by FPGAs, have developed cloud- and edge-tailored devices, promoting the integration of FPGAs at all layers [Leeser'21, Xu'22].

Despite the benefits that FPGAs offer in this environment, challenges still need to be addressed for FPGAs to be effectively deployed in the cloud-edge continuum. The primary premise of the cloud-edge continuum is that the computing resources comprising the continuum are viewed from the user's perspective as a unified entity responsible for executing part of the workload. Then, based on the requirements of each application (e.g., performance or energy constraints) and the state of the computing resources of the continuum (e.g., availability, computing power, power consumption), these applications should be mapped to specific computing resources transparently to the user. However, two main challenges appear: First, FPGAs need to be virtualized in the continuum, in the sense that applications can be accelerated in any node, regardless of the underlying hardware available in each node. Second, there is a need for an optimal workload management strategy that efficiently maps applications to specific computing resources based on run-time constraints.

Virtualization is an open issue particularly challenging on FPGAs where direct control of the underlying hardware (e.g., DPR, data transference) requires low-level platform-specific architectural knowledge to render optimal results [Bobda'22]. To address this challenge, cloud providers such as Amazon Web Service (AWS) offer users access to dedicated FPGA nodes through a Platform-as-a-Service (PaaS) model.[1] Such an approach serves as a workaround that results in an infrastructure that does not scale well and is complex to manage due to its monolithic service model. This strategy shifts the responsibility to the users, who must design and manage ad-hoc applications tailored to the specific hardware of the FPGA node, thereby undermining the notion of a computing continuum in which resource details remain transparent to the user. Similarly, resource management is another major challenge for FPGAs. The computing resources of the continuum typically face a highly dynamic scenario where the applications to be accelerated may arrive at any point in time. Hence, workloads can no longer be expressed as known static task graphs that can be conveniently scheduled at the design phase, but as dynamic workloads that may evolve over time.

[1] https://aws.amazon.com/ec2/instance-types/f2/

Therefore, in order to execute these dynamic workloads efficiently on FPGAs, there is a need for dynamic resource management that, for a given constraint (e.g., performance goal, energy budget), performs local optimizations by taking informed scheduling decisions at run time based on current system information (i.e., application and node-specific details) that are subject to dynamic changes.

1.2 Main Goals of the Thesis

To enable the efficient use of reconfigurable systems in the cloud-edge continuum and solve the FPGAs virtualization and resource management challenges described in Section 1.1, this Thesis establishes a series of goals that need to be achieved:

1. Design an infrastructure capable of handling and accelerating dynamic workloads by deploying hardware accelerators in reconfigurable multi-accelerator computing resources. The infrastructure must be agnostic to the underlying hardware to comply with the cloud-edge continuum.

2. Research on novel techniques to support the seamless movement of hardware-accelerated applications across the different nodes of the cloud-edge continuum transparently. Users should not be required to know low-level details of the reconfigurable nodes present in the continuum.

3. Propose and develop a methodology and associated framework to monitor at run time the execution of dynamic workloads that can be later leveraged to perform local optimizations (i.e., scheduling decisions that render performance or energy improvements at the node level). Power consumption and execution time are the key targets to be monitored using this framework.

4. Investigate novel methodologies to characterize the execution of reconfigurable multi-accelerator systems from run-time operational measurements gathered by the monitoring framework. The characterization must adapt to changes (e.g., variations in the applications that comprise the workload or the system's capabilities) that may occur over time.

5. Research and implement a scheduling strategy that optimizes the execution of dynamic workload on reconfigurable multi-accelerator systems. This strategy must adhere to specific goals (i.e., performance or energy) and base its optimization on the previously mentioned run-time characterization.

6. Design and evaluate validation campaigns that assess the developed technology per component and from end to end. A combination of benchmarks and

real-world use-case scenarios is required to stress the technology and analyze capabilities such as portability, scalability or energy efficiency.

1.3 Research Context: The MYRTUS Project

Most of the work in this Thesis has been developed under the umbrella of a Horizon Europe research project called MYRTUS (granted in the HORIZON-CL4-2023-DATA-01-04 topic).[2] To put into perspective how the work developed in this Thesis can be transferred to the industry, this section briefly describes the motivation behind the MYRTUS project, its main objectives, and how the work proposed in this Thesis contributes to achieving these objectives.

Figure 1-1: MYRTUS logo.

Motivation

Recent world-changing events, such as the COVID-19 pandemic and extreme weather events related to climate change, have driven significant investments in the digitalization and virtualization of computing systems over the past few years. Ubiquitous computing and the concept of smart everything everywhere have been implemented to enable telematic work and remote management of industrial sectors, resulting in a vast amount of data that must be processed.[3] Moreover, applications present different requirements, from real-time constraints where data has to be processed close to its source, to high-performance needs requiring computation in cloud facilities. Therefore, similarly to the goals of this Thesis (see Section 1.2), the MYRTUS project envisions that novel approaches are required to design and manage such a complex computing environment, combining cloud, fog, and edge platforms into a unique computing entity that provides real-time, scalable, sustainable, and secure processing capabilities.

[2] https://myrtus-project.eu/
[3] https://smartanythingeverywhere.eu/

In addition, the increase in digitalization and virtualization has boosted the human-to-computer interaction in fields like industry 5.0, which goes towards the idea of humans cooperating with machines.[4] Cyber Physical Systems (CPS's), which tightly integrate computation, networking and physical processes, are now moving to a living dimension, enabling cooperation with humans, powering the concept of autonomous systems. Hence, the development of responsible and sustainable computing, guided by Artificial Intelligence (AI) strategies, is mandatory to drive the new generation of CPS's that will be present in the cloud-edge continuum.

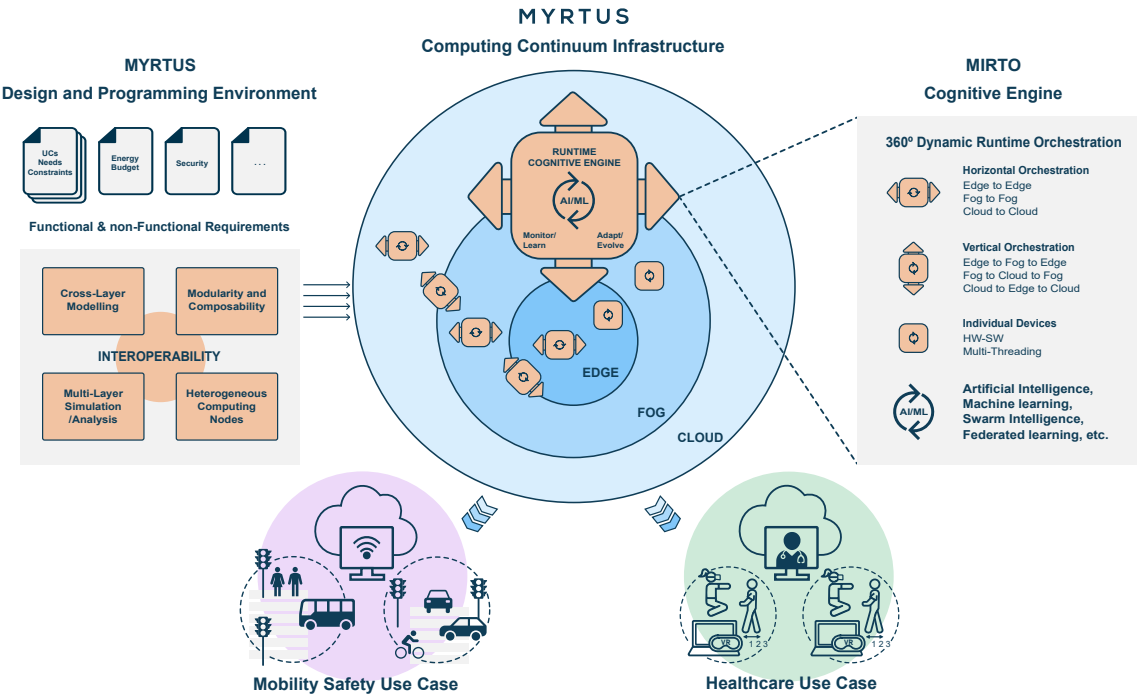


Figure 1-2: MYRTUS general overview [Palumbo'24].

Objectives

MYRTUS [Palumbo'24] is intended to provide a technology for integrating edge, fog, and cloud computing into a seamless execution environment, redesigning tools to orchestrate collaborative and distributed components. The goal is to build an open, secure, trustworthy, and sustainable computing ecosystem and provide these results:

1. **A Reference Infrastructure for the Cloud-Edge Continuum:** a composable multi-layer infrastructure that integrates heterogeneous and collaborative computing nodes across the cloud, fog and edge. The goal is to provide a trustworthy, energy-efficient, yet powerful computing capacity.

[4] https://industry4o.com/2022/05/02/industry-5-0-a-human-centric-smart-manufacturing/

2. **A Run-Time Orchestration Cognitive Engine:** it manages the resources of the continuum to maximize performance, energy efficiency and scalability, while providing security, privacy, and trust. Adaptive mechanisms, such as workload optimization strategies, enable the different layers of the continuum to adapt autonomously to internal and external stimuli, utilizing AI-based strategies to facilitate self-organization and self-management of heterogeneous resources.

3. **A Design and Programming Environment (DPE):** it supports the operation of complex computing infrastructures composed of distributed heterogeneous computing devices. It enables i) modeling, simulation, and analysis of the continuum at the system level, ii) model to implementation transition, and iii) node-level optimization and deployment, fostering interoperability.

4. **Validation in Real-World Use Cases:** a healthcare use case will use the MYRTUS technology to enable scalable collaborative tele-rehabilitation for cognitive and neuromotor purposes, using visuo-haptic interactions between patients in virtual reality scenarios. A mobility use case will explore traffic safety management at road intersections by leveraging cooperation between autonomous vehicles, road users' personal devices, and traffic information.

Alignment with the Thesis

The work developed in this Thesis, introduced in Section 1.2, covers the requirements of the MYRTUS project that are specific to FPGAs and novel reconfigurable edge nodes:

- The infrastructure to manage dynamic hardware-accelerated workloads in FPGAs, designed to be compatible with any platform of the cloud-edge continuum (i.e., Thesis goal #1). Together with the support for seamless movement of applications across the nodes in a user-agnostic manner (i.e., Thesis goal #2), it provides the technology required to build the edge-level reference infrastructure of MYRTUS on the FPGA side (i.e., MYRTUS goal #1).

- The monitoring framework (embedded in the infrastructure) extracts run-time data from the dynamic workloads (i.e., Thesis goal #3), which, thanks to the characterization methodology proposed, enables deriving insights from the system (i.e., Thesis goal #4). This information is then leveraged to guide a scheduling strategy to perform run-time optimizations at the node level (i.e., Thesis goal #5), enabling the MYRTUS orchestration engine to perform node-level optimizations on the FPGA nodes of the continuum (i.e., MYRTUS goal #2).

MYRTUS, however, is a large multi-partner project, and the work developed in this Thesis only covers some of the duties committed to the Universidad Politécnica de Madrid as a project partner. In the reference infrastructure, the Università degli Studi di Cagliari and the Università degli Studi di Sassari are the partners that complement

the edge layer with Coarse-Grained Reconfigurable Array (CGRA)-based devices, while HIRO contributes a compact micro data center for the fog tier, and Abinsula supplies a multi-sensor gateway. All of these components are seamlessly integrated via the Liqo multi-cluster manager developed by ArubaKube. At the orchestration level, the node-level run-time optimizations of the MYRTUS cognitive engine presented in this Thesis are complemented by continuum-wide decision-making contributions from other consortium partners. The MYRTUS use cases introduce a level of complexity that this Thesis cannot fully address; therefore, simpler validation scenarios have been elaborated to evaluate the technology explicitly developed in the Thesis. The implementation and evolution of the DPE is distributed across multiple organizations, and every member of the consortium actively collaborates in validating the MYRTUS technology through the use cases.

1.4 Technology Background

This section presents the technological framework around the Thesis. Therefore, it covers the basic concepts of reconfigurable computing, the cloud-edge continuum, and task scheduling.

1.4.1 Reconfigurable Computing

Traditionally, computation followed two distinct paths. One option was to perform computation purely in hardware, mainly using application-specific circuits (i.e., ASICs). ASICs are designed to perform a specific computation, resulting in a highly efficient solution. However, since the hardware cannot be modified after its construction, a complete re-manufacturing must be undergone to support a different computation, which is highly costly due to the associated non-recurrent engineering costs [Khazraee'17]. The alternative was to use software-programmable microprocessors. In this case, programmers must express the target application as a set of instructions to be executed sequentially. This approach offers greater flexibility, as changing the software program is sufficient to adapt the algorithm performed by the microprocessor. However, the classic Von Neumann model, which fetches, decodes, and executes each instruction from a shared memory, introduces a performance bottleneck. In addition, the required fetch/decode pipeline incurs an energy overhead compared to a dedicated datapath tailored to the target computation.

Reconfigurable computing bridges the gap between software and hardware. The rationale behind this technology is to create devices that, through a process known as reconfiguration, can adapt their processing datapath (i.e., their functionality) either at design or run time. This provides reconfigurable computing systems with higher

performance than software alternatives while maintaining a higher degree of flexibility compared to hardware [Compton'02]. Unlike processors, which rely on temporally distributed computation, reconfigurable devices preserve the principle of spatially distributed computation, similarly to ASICs.

Reconfigurable Devices

Reconfigurable devices are the physical hardware components that enable adaptive functionality through reconfiguration. They differ in how fine their resources are partitioned for reconfiguration, and hence can be classified based on their reconfiguration granularity:

- **Fine-grained Architectures:** this category includes devices that present basic logic elements, such as Look-Up Tables (LUTs) and Flip-Flops (FFs), which allow them to be configured at the bit level. The most prominent exponents of this category are FPGAs, which has been established as the de facto industrial leader for supporting reconfigurable computing since the late 1980s.
- **Coarse-grained Architecture:** this category includes devices with more complex logic elements, such as Arithmetic Logic Units (ALUs), which allow for configuration at the word level. These devices, known as CGRAs, present their Processing Elements (PEs) distributed in an array interconnected following a particular topology (typically a 2D mesh). Some examples of 2D mesh CGRAs are ADRES [Mei'04] from IMEC, or MorphoSys [Singh'00] from the University of California, Irvine. Compared to fine-grained devices, CGRAs present shorter reconfiguration times, but they offer less flexibility.

The above classification refers to the physical reconfigurable devices themselves. To bridge the gap between these low-level fabrics and higher-level programming models, the overlay concept appeared as a virtual reconfigurable architecture mapped on top of a commercial FPGA fabric. Overlays typically expose a coarse-grained architecture (e.g., CGRA-style processing elements and interconnections) and provide a software-friendly Application Programming Interface (API), enabling reconfiguration at the overlay level rather than regenerating an entire FPGA bitstream. By abstracting low-level details, overlays combine the performance advantage of hardware specialization with the flexibility of software.

A generic FPGA architecture is shown in Figure 1-3. The design employs an island-based layout, the most commonly used architecture in modern FPGAs [Gandhare'19], where reconfigurable logic resources are organized into islands surrounded by programmable interconnect channels and Switch Boxes (SBs), enabling efficient signal routing and modular scalability. In this basic and manufacturer-agnostic model, Configurable Logic Blocks (CLBs) contain the basic logic blocks mentioned before (i.e., LUTs and FFs), dedicated on-chip memory blocks are introduced for storage and

Digital Signal Processor (DSP) blocks for optimized arithmetic operations. A network of horizontal and vertical pre-fabricated wires connects all the logic resources in the device. The network routing is configured with SBs and the logic is connected through Connection Boxes (CBs).

FPGAs can be regarded as two-layer devices: a programmable hardware logic layer containing the reconfigurable resources (e.g., LUTs, FFs), and a configuration memory layer whose contents define the functionality of each resource. Customization is achieved through a bitstream configuration file, which, once loaded into the memory layer, programs the logic fabric and enables rapid reconfiguration without altering the physical hardware.

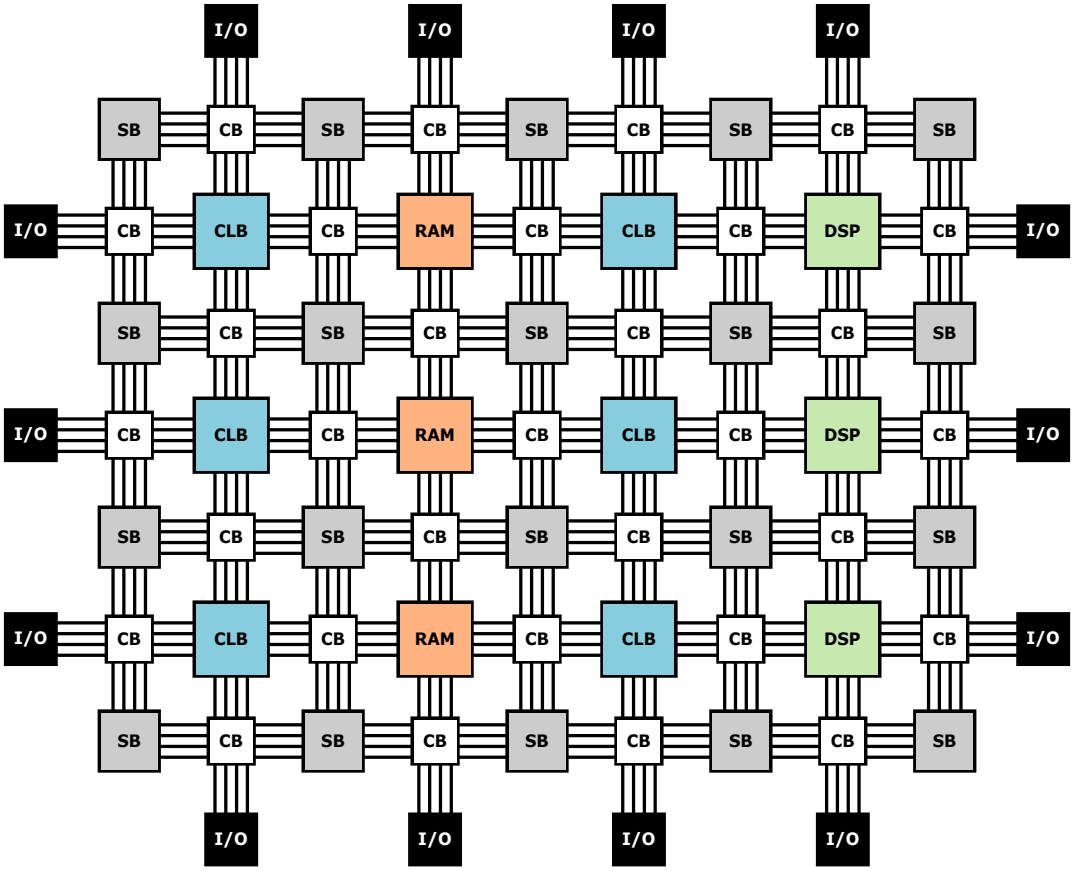


Figure 1-3: General architecture of an island-based FPGA.

System on Programmable Chip

The standard approach to integrating software and hardware components into a heterogeneous entity is to build an ASIC that combines multiple computing components on the same chip, which is known as a System on Chip (SoC) [Brackenbury'10]. When this concept is implemented on the reconfigurable fabric of an FPGA, the resulting system is referred to as a System on Programmable Chip (SoPC). SoPC architectures typically integrate reconfigurable accelerators alongside general-purpose Central Processing Units (CPUs) to leverage the best of both worlds. In these architectures, the reconfigurable fabric typically handles compute-intensive, massively parallel kernels, while the embedded processor efficiently executes inherently sequential tasks such as control flow, conditional branches and variable-length loops.

Reconfigurable accelerators integrated with SoPC can be classified depending on their coupling with the embedded microprocessor [Compton'02] (see Figure 1-4):

- **Functional Unit (FU):** the reconfigurable accelerator is included in the datapath of the microprocessor. This allows the extension of the Instruction Set Architecture (ISA) of the microprocessor, enabling custom instructions that can change over time.
- **Coprocessor:** the reconfigurable devices can perform computations without supervision from the microprocessor, which must only initialize the reconfigurable accelerator, provide input data, and retrieve the results. Hence, both the microprocessor and the coprocessor can perform simultaneous computations.
- **Attached Processing Unit:** the reconfigurable partition is treated as an additional processing unit in a multi-processor system. Hence, communication between them must be conducted via specialized processes. This approach gives the reconfigurable accelerator autonomy to perform significant sections of computation.
- **Standalone Processing Unit:** the reconfigurable accelerator acts as a standalone processing unit, performing extensive periods of computation without the intervention of a companion microprocessor (which might not even be present).

To ease the development of SoPC architectures, Intellectual Property (IP) libraries are available for designers. In this context, IP cores are pre-designed hardware modules that can be licensed and integrated into FPGA or ASIC designs. The IPs used could be either hard-core components (i.e., physical implementations) or soft-core components (i.e., Register-Transfer Level (RTL) descriptions).

Please note that some of the most successful SoPC-based FPGA devices on the market are the Zynq and Zynq UltraScale+ families from AMD Xilinx, which are the selected candidates for the experimental validation of this Thesis. In these devices, reconfigurable logic is referred to as Programmable Logic (PL), and hard-core

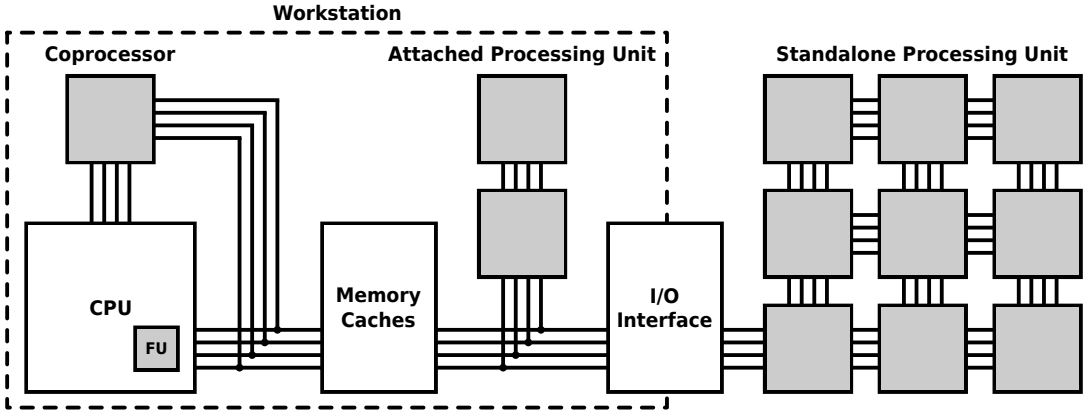


Figure 1-4: Coupling levels in a reconfigurable system [Compton'02].

components (in this case, ARM processors) are known as Processing System (PS). In this way, the PL executes the reconfigurable accelerators (or, in general, the hardware IPs), while the PS executes the software of the application.

Dynamic and Partial Reconfiguration

FPGAs can be classified depending on the memory technology used to store its configuration. As stated in [Azarian'09], SRAM-based FPGAs present certain advantages, such as higher logic resource density and performance compared to alternative technologies (e.g., flash-based FPGAs). On the other hand, some of their main disadvantages include higher power consumption and the need to reprogram the logic after each power cycle. However, the fundamental feature of SRAM-based FPGAs is that they can be reconfigured indefinitely, which gives them higher flexibility.

In turn, the reconfiguration process can be classified based on various factors [Azarian'09]. A first taxonomy can be made based on the amount of resources involved in the process:

- **Full reconfiguration:** it involves the reconfiguration of all logic resources of the FPGA fabric.
- **Partial reconfiguration:** it affects only part of the logic resources of the FPGA (i.e., a reconfigurable region).

An additional classification can be extracted from the way the reconfiguration process interferes with the execution of the rest of the system:

- **Static reconfiguration:** during the reconfiguration process, the entire system is stopped. It is typically performed before the application execution.
- **Dynamic reconfiguration:** during the reconfiguration process, while a region of the logic is reconfigured, the rest of the system remains in operation. It is typically performed at run time, parallel to the application execution.

Different options from both taxonomies can be combined. The standard FPGA reconfiguration, which happens at power up, is static and full. A less common alternative is static and partial reconfiguration, where only a subset of the device is configured at startup, progressively activating additional regions as needed. This approach can reduce both configuration time and energy consumption during the power-up process [Lombardo'12]. Nevertheless, what gives FPGAs their software-like flexibility and hardware-like performance is the use of DPR [Koch'12], which consists of reconfiguring a region of the FPGA logic at run time while the rest of the system remains in operation.

ARTICo^3

Offloading applications into the logic fabric of the FPGA involves multiple steps (e.g., reconfiguration, data transfer, execution monitoring) that must be carefully orchestrated. This process gets even more complex when numerous application executions must be managed simultaneously. A convenient approach to handling such a scenario in a manner that allows for efficient and scalable use of FPGA resources is to rely on a framework that eases the implementation of reconfigurable hardware-accelerated systems. Every component developed in this Thesis has been designed to be agnostic of the underlying hardware acceleration framework (as will be demonstrated throughout the document). However, since it is impractical to demonstrate the proposed technology in every hardware acceleration framework, a representative option must be selected. Therefore, among the options available from industry and academia, ARTICo^3 [Rodríguez'18] is being selected as the framework for handling hardware acceleration implementation in this Thesis. Please refer to Section 2.2.1 for more details on the reasoning behind its selection.

ARTICo^3 is an open-source framework targeted for high-performance embedded computing scenarios. It performs run-time hardware acceleration, offering adaptive computing performance, energy efficiency, and fault tolerance on demand.

The framework, represented in Figure 1-5, consists of the following components:

- **Reconfigurable processing architecture:** the FPGA fabric is divided into reconfigurable slots. The slot-based partitioning, combined with DPR capabilities, allows for a multi-accelerator hardware implementation approach where the hardware accelerators behave as attached processing units [Compton'02], offering both high flexibility and performance. ARTICo^3 employs a Single Instruction, Multiple Threads (SIMT) computing model, an eminently parallel model of computation where multiple replicas of a hardware accelerator can be configured simultaneously (i.e., in more than one ARTICo^3 slot) to enable performance scalability and energy efficiency trade-off exploration. It also adds a layer of abstraction between the user and the underlying hardware, performing data distribution to the hardware accelerators with no user intervention.

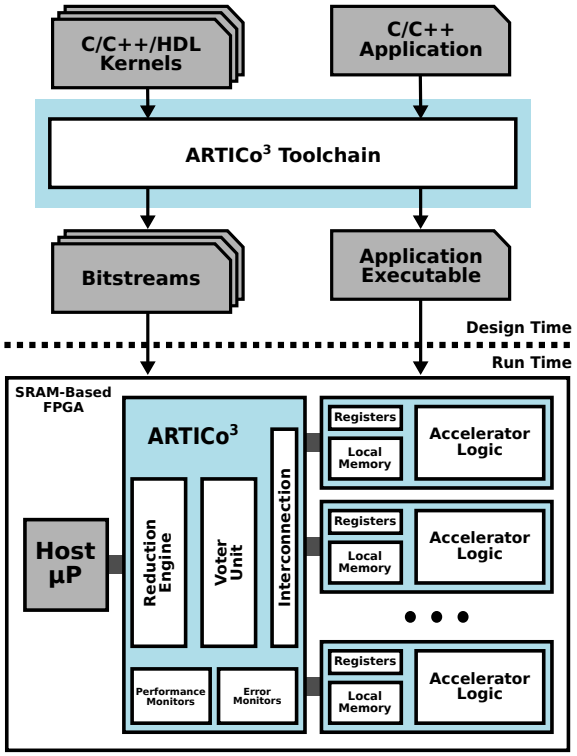


Figure 1-5: Overview of the ARTICo^3 framework [Rodríguez'20].

- **Runtime library:** user application and hardware are separate entities. The runtime library offers an API to perform the operations required to accelerate applications (e.g., system initialization, accelerator loading, data management). The runtime handles low-level tasks, such as the hardware reconfiguration process and Direct Memory Access (DMA) operations, to transfer data to/from the accelerators.

- **Automated toolchain:** it enables the user to design a reconfigurable system from a host application and the description of the kernels (i.e., functions to be accelerated in hardware). The users provide the host applications in $C/C++$ code and the kernel descriptions in Hardware Description Language (HDL) or $C/C++$ descriptions compatible with High-Level Synthesis (HLS) tools.

1.4.2 Cloud-Edge Continuum

As introduced in Section 1.1, the cloud-edge continuum is a relatively new computing paradigm [Carmo'17] that leverages the cloud's virtually infinite computing resources and the edge's versatility and availability, resulting in a unified space of computing resources known as the continuum. This continuum presents the computing resources (also referred to as nodes) distributed in a layered manner, with boundaries between the layers that are loose and flexible.

The cloud-edge continuum is a broad field of study that merges the previously described concepts of cloud, fog, and edge computing. However, this section will cover just the fundamental aspects of the cloud-edge continuum that motivated the work developed in this Thesis and are, therefore, required for its proper understanding.

Dynamic Workloads

The high flexibility provided by the layered structure of the cloud-edge continuum makes it a computing infrastructure suitable for a diverse set of workloads that may support a wide range of different applications. Such workloads can range from performance-hungry applications typically executed in the cloud to privacy-sensitive or latency-limited applications usually deployed at the edge, as well as all the applications with mixed requirements that may fall in between.

Moreover, the continuum is treated as a unique entity where the applications to be executed are received and then mapped to the computing resources. Run-time application requirements dictate this mapping and the current continuum environment (e.g., available resources, performance capabilities of each node) [Khalyeyev'23]. From the perspective of the computing resources within the continuum, these are, therefore, dynamic workloads that are significantly challenging to execute efficiently.

Virtualization Techniques

Another key point of this computing paradigm is that users must be agnostic of the underlying hardware. Once an application is submitted to the continuum for execution, it will be offloaded to the node that best fits its requirements without user notice or intervention. This convenient and powerful behaviour poses a difficult challenge: portability. Applications should be able to run on any node of the continuum, regardless of the specific computing platform underneath. To tackle that, the following virtualization techniques are typically used for application deployment:

- **Containerization:** consists of encapsulating an application and all its dependencies (e.g., libraries, tools) into a lightweight and portable image. What is packed inside a container is isolated from the rest of the system by design. The concept of containerization is built with portability and scalability as its foundational blocks. The software used for creating and managing containers is known as a container runtime (e.g., Docker.[5], CRI-O[6]). In fact, there is an open standard container format, known as Open Container Iniciative (OCI), which dictates how containers should be created and managed; most container runtimes adhere to it.[7] Therefore, most container images can be run with any

[5]https://www.docker.com/
[6]https://cri-o.io/
[7]https://opencontainers.org/

other runtime alternative, which is highly convenient for deployments in the cloud-edge continuum.

While traditional virtualization techniques (i.e., virtual machines such as VMWare or VirtualBox) place the whole Operating Systems (OS's) stack on top of the host, containerization interacts directly with the host OS, resulting in a more lightweight solution with near-native performance (see Figure 1-6) [Bentaleb'22].

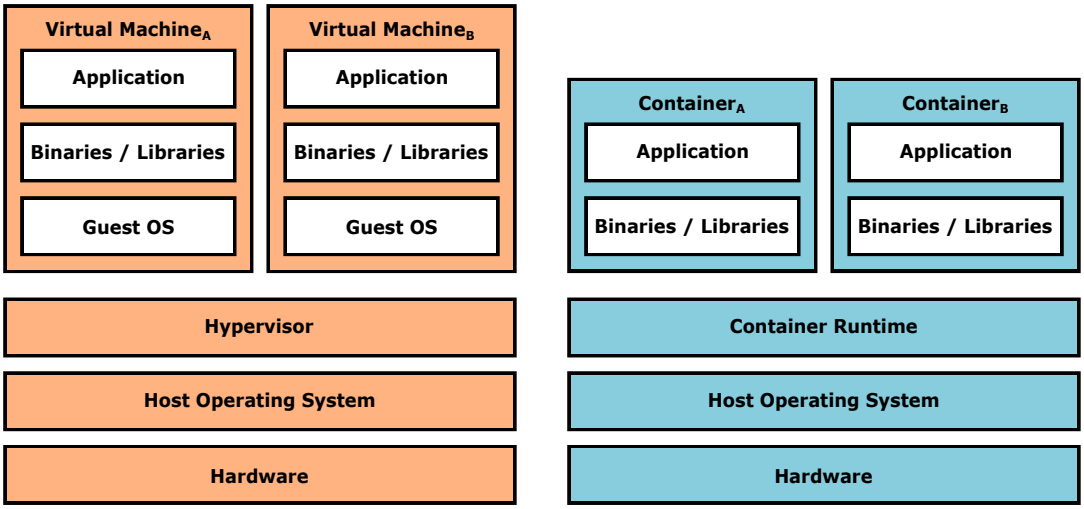


Figure 1-6: Virtual Machines (left) and Containers (right) components.

- **Orchestration:** containers are efficient and convenient solutions for application deployment. However, handling multiple containers simultaneously can become increasingly challenging for a large computing infrastructure. Each container executed on a node must be deployed individually, along with any other companion resources (e.g., shared storage, network service). This approach does not scale well and is prone to errors. To address such a limitation, the concept of container orchestration emerges. It automates the process of handling the containers transparently for the user. It deploys containers, allocates resources, handles storage, and manages networking, while granting application scalability and failover [Jawarneh'19]. The most commonly used orchestration tools are Kubernetes[8] and Docker Swarm.[9]

[8]https://kubernetes.io/
[9]https://docs.docker.com/engine/swarm/

1.4.3 Scheduling

A significant degree of flexibility can be achieved when implementing hardware acceleration in FPGAs, specifically by following a multi-accelerator approach. This approach involves a single device hosting multiple accelerators over time, replicating them to increase throughput, or changing its configuration to suit the needs of a dynamic workload.[10] In such reconfigurable multi-accelerator systems, a slot-based partitioning (as ARTICo^3 provides) comes out as a convenient solution to couple FPGA together with their DPR capabilities to enable the exploitation of both task-level parallelism, by running multiple kernels in parallel, and data-level parallelism, by implementing various replicas of the same kernel (see Figure 1-7). Indeed, a combination of both can be explored by configuring multiple kernels in the FPGA fabric, each with a certain number of replicas (limited by the amount of available reconfigurable slots on the device). However, deciding on the specific configuration (i.e., the combination of kernels and the number of replicas per kernel) to be implemented in the FPGA also poses a challenge [Diessel'01]. Given certain constraints, such as deadlines or power budget, the user must choose the best configuration to implement among all possible ones, as well as the specific moment at which it should be configured, which is referred to as a scheduling problem.

Scheduling falls into the field of combinatorial optimization problems, as the search space consists of a finite and discrete set of solutions resulting from the possible combinations of assigning tasks to resources. The goal is to find the solution that optimizes an objective function (i.e., a function that evaluates how good a solution is, based on particular scheduling goals) among this vast number of alternatives [Papadimitriou'98]. The challenge of scheduling on FPGAs is that the number of available configurations (i.e., solution space) grows in a factorial manner with the number of kernels (and reconfigurable slots) available [Adhi'18]. The problem of task scheduling in FPGAs is an NP-Hard problem [Blazewicz'83], which, leaving aside other mathematical implications, means that, in practice, the optimal solution cannot be found efficiently in terms of computation time.

Since the solution space can be massive, the best approach to solving these optimization problems is to employ algorithms that reduce the number of solutions to be evaluated before making a decision. Heuristics and metaheuristics are the most adopted approaches, generating near-optimal solutions rather than optimal solutions that can be efficiently computed [Yang'10].

The key points of each of them are the following [Houssein'21]:

- **Heuristics:** are problem-dependent algorithms requiring specific application-domain knowledge. They are typically based on greedy strategies, rule-based decisions, or precomputed policies that are computationally efficient and aim for satisfactory solutions rather than global optimal solutions. This approach is

[10] The scheduling problem tackled in this Thesis focuses on reconfigurable multi-accelerator systems and the specific constraints they introduce.

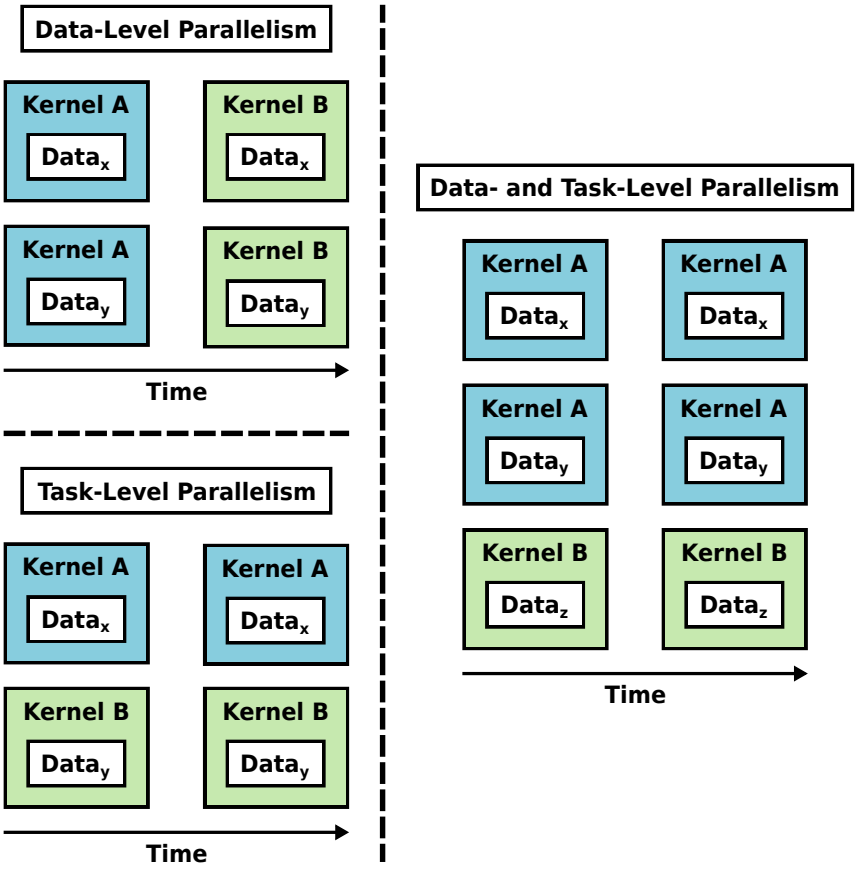


Figure 1-7: Representation of data-level and task-level parallelisms. In data-level parallelism, multiple instances of the same kernel compute different data simultaneously. In task-level parallelism, different kernels execute in parallel over distinct data sets. Data- and task-level parallelism combines both ideas by running different kernels in parallel, some of which have multiple replicas. Note that each distinct kernel is represented by a color.

advantageous in static (i.e., tasks and available resources known at design time) and moderately complex scheduling problems [Houssein'21].

- **Metaheuristics:** are problem-independent, higher-level strategies that guide the solution search process in an adaptive or stochastic manner, allowing for better solution space exploration and attempting to avoid local optima. Metaheuristics (e.g., inspired by physical phenomena, evolutionary algorithms, or swarm intelligence) can dynamically adjust their search behavior in response to run-time variations in workloads and system conditions. This flexibility and adaptability make metaheuristics particularly well suited for highly complex and dynamic scheduling problems, where the search space and objectives may change unpredictably [Houssein'21].

Choosing between heuristics and metaheuristics as the mathematical solver of the scheduling problem will depend on the complexity and dimensions of the problem

to be solved, how reasonable a near-optimal solution is compared to the optimal one, and the amount of time available for searching options within the solution space (which will depend on the timing constraints of the targeted application). In general, heuristics are preferable for static and moderately complex problems, where efficient and straightforward rules can yield satisfactory results. Conversely, metaheuristics become more advantageous in highly complex and dynamic scenarios, as their stochastic and adaptive search strategies provide greater flexibility in dealing with run-time variations in workloads and system conditions. Moreover, within heuristics and metaheuristics, there are different flavors explicitly tailored to meet particular scheduling objectives (e.g., adaptability, convergence time) [Houssein'21].

1.5 Thesis Overview and Layout

The technical work developed in this Thesis is divided into different components that operate together to address the proposed goals. This section provides a brief overview of the components and their interactions. Later, the structure of this document, along with a brief description of each chapter, is presented.

1.5.1 Overview

An overview of the technical work presented in this Thesis is shown in Figure 1-8. It is structured in three main pillars: an infrastructure to manage resources in FPGA-based reconfigurable multi-accelerator systems in the context of the cloud-edge continuum, a data-driven methodology to model these systems during operation, and a scheduling strategy to optimize the workload deployment at run time.

The following components cooperate to achieve the overall goal of optimizing the acceleration of dynamic workloads in the cloud-edge continuum:

Infrastructure (in blue, in Figure 1-8)

When a dynamic workload needs to be accelerated, the proposed infrastructure manages the arrival of each task (i.e., hardware acceleration requests) by storing them in the waiting queue shown in Figure 1-8. These tasks are then selected for hardware offloading following a specific scheduling policy. Afterwards, they are deployed in the fabric of the FPGA as hardware accelerators, following the procedure illustrated by the block named task container in Figure 1-8 (the corresponding partial bitstreams must be made available beforehand). The process continues until the dynamic workload is fully executed. In the meantime, a companion monitoring infrastructure

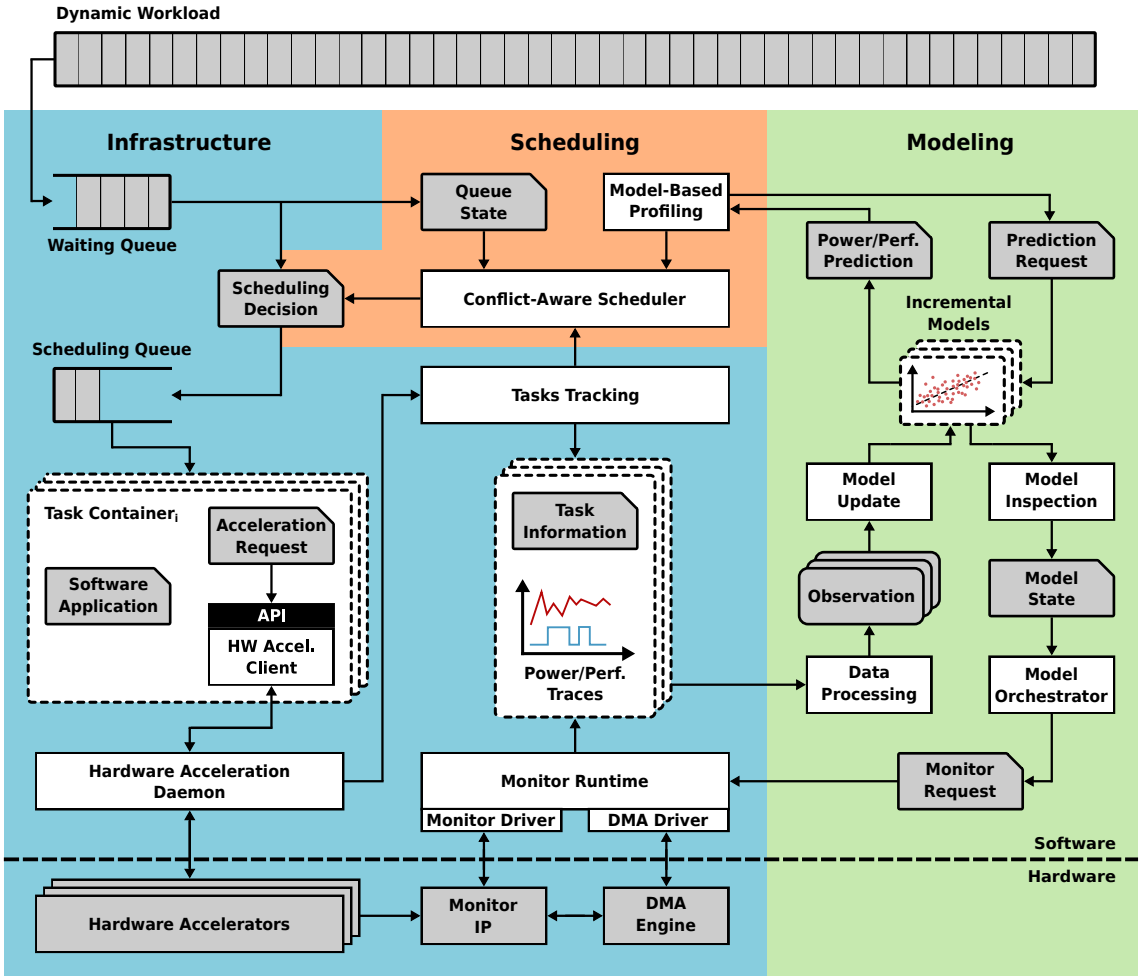


Figure 1-8: Thesis technical work overview.

extracts, at run time, synchronized power and performance traces from the hardware accelerators under execution in the FPGA. The proposed infrastructure is platform-agnostic and therefore compatible with the nodes of the cloud-edge continuum that support hardware acceleration.

Modeling (in green, in Figure 1-8)

On the right side of Figure 1-8, it is shown how the traces obtained from the hardware acceleration are processed by the components associated with this modeling pillar, and then used to train Machine Learning (ML) models for predicting power consumption and performance. These models are kept up to date by tracking their prediction accuracy in real time, retraining them incrementally with new data as needed. A dedicated model training process, referred to as model orchestration in Figure 1-8, has been designed to keep models updated throughout the entire system's lifetime while inducing minimal overhead.

Scheduling (in orange, in Figure 1-8)

The decision of which specific combinations of tasks and their number of replicas are configured in the FPGA fabric has a significant impact on the performance and power consumption of the system (an in-depth explanation of this phenomenon is presented in Section 3.3). As a result, a run-time profiling of the system based on the predictions of the models mentioned above, together with specific information about the tasks to be accelerated, is leveraged to guide task scheduling decisions that optimize the execution of the dynamic workload.

1.5.2 Document Layout

The document consists of six chapters. This first chapter provides an introduction, followed by chapters dedicated to each of the three components of the technical development of this Thesis. The integration of those components in two use cases constitutes the fifth chapter, followed by the last chapter that concludes the document.

Chapter 1 - Introduction

This chapter. The motivation behind the work developed in this Thesis is presented first. Next, the main objectives are highlighted, followed by a description of the research context of this Thesis. Another section briefly introduces the technology background required to understand the rest of the document. Finally, this section provides an overview of the Thesis's work and its structure.

Chapter 2 - Resource Management Infrastructure for the Cloud-Edge Continuum

The second chapter is devoted to the proposed resource management infrastructure. After analyzing the state of the art, the infrastructure components are described at the node and cloud-edge continuum levels. The chapter ends with a section that performs a validation campaign for each constituent of the infrastructure.

Chapter 3 - Data-Driven Modeling of Dynamic Workloads

The third chapter presents the data-driven modeling strategy. An analysis of kernel interaction motivates the modeling. Next, an analysis of the modeling strategies found in the literature is given, followed by a description of the proposed approach. The next step is to extend this modeling strategy to support incremental updates. The chapter concludes with a validation section.

Chapter 4 - Scheduling for Optimal Resource Management

The fourth chapter focuses on a scheduling mechanism to optimize resource management. The description and implementation of the proposed conflict-aware scheduler follow the analysis of state-of-the-art scheduling approaches. A final validation section evaluates the proposal.

Chapter 5 - Component Integration

The fifth chapter shows the integration of the three aforementioned components into two real use cases. A description of these use cases is presented before detailing the actual integration of the components of this Thesis. To conclude, this chapter will present experimental results demonstrating the performance of the integrated system, where all the technical pillars developed in this Thesis come together.

Chapter 6 - Conclusions, Impact and Future Lines of Work

The final chapter. It highlights the conclusions drawn from the work of this Thesis, along with a summary of its main contributions. An additional section points out the impact of the work presented in this Thesis on publications, research collaborations, co-supervised works, and research projects. To conclude, the possible future research directions coming from this Thesis are discussed.

Chapter

2 RESOURCE MANAGEMENT INFRASTRUCTURE FOR THE CLOUD-EDGE CONTINUUM

This chapter presents the infrastructure proposed in this Thesis for managing resources in FPGA-based reconfigurable multi-accelerator systems. The proposed infrastructure targets cloud-edge continuum environments, enabling the orchestration, deployment, and monitoring of hardware-accelerated dynamic workloads in a platform-agnostic manner.

The chapter begins with a brief overview of the main components of the proposed infrastructure, followed by a section that analyzes the most relevant state-of-the-art work. The technical development of the infrastructure is described next, at two subsequent levels: the node-level infrastructure and its extension to the cloud-edge continuum. The chapter concludes with the validation campaign carried out to evaluate each infrastructure component using well-known benchmarks for hardware-accelerated computing.

2.1 Chapter Overview

When FPGAs are used to accelerate the dynamic workloads present in cloud-edge continuum scenarios, a series of challenges have to be addressed. First, assuming that in real-world scenarios the tasks to be accelerated arrive at the device in an uncontrolled manner, a reactive mechanism needs to attend to those tasks and offload them efficiently into the FPGA fabric in a continuous manner. This mechanism will rely on the run-time optimizations supported at the hardware level by the DPR capabilities of the infrastructure underneath. Additionally, a non-intrusive monitoring mechanism should be available and aligned with the offloading mechanism to enable the acquisition of relevant traces during system operation, a fundamental requirement for achieving the workload characterization and optimization mentioned as part of the goals of this Thesis.

Furthermore, since the hardware beneath the devices that form the continuum could be completely different, these mechanisms must be modular and portable, ensuring their proper operation on the target device. Similarly, since tasks on the

cloud-edge continuum can be located in any of the available nodes with zero user intervention, these tasks must be fully portable across different devices, requiring a certain degree of virtualization.

The infrastructure proposed in this Thesis was designed to address these precise challenges. Leveraging the ARTICo^3 framework, it manages the hardware acceleration of dynamic workloads (composed of hardware acceleration requests) in FPGA-based reconfigurable multi-accelerator systems while performing run-time power consumption and performance trace acquisition, independent of the underlying hardware, enabling its implementation in any node of a cloud-edge continuum scheme. Please note that, even though ARTICo^3 is used for hardware acceleration implementation, all components are framework-agnostic and the approach is compatible with (and generalizable to) other hardware acceleration frameworks. Figure 2-1 highlights the main components of the proposed infrastructure above those of the rest of this Thesis.

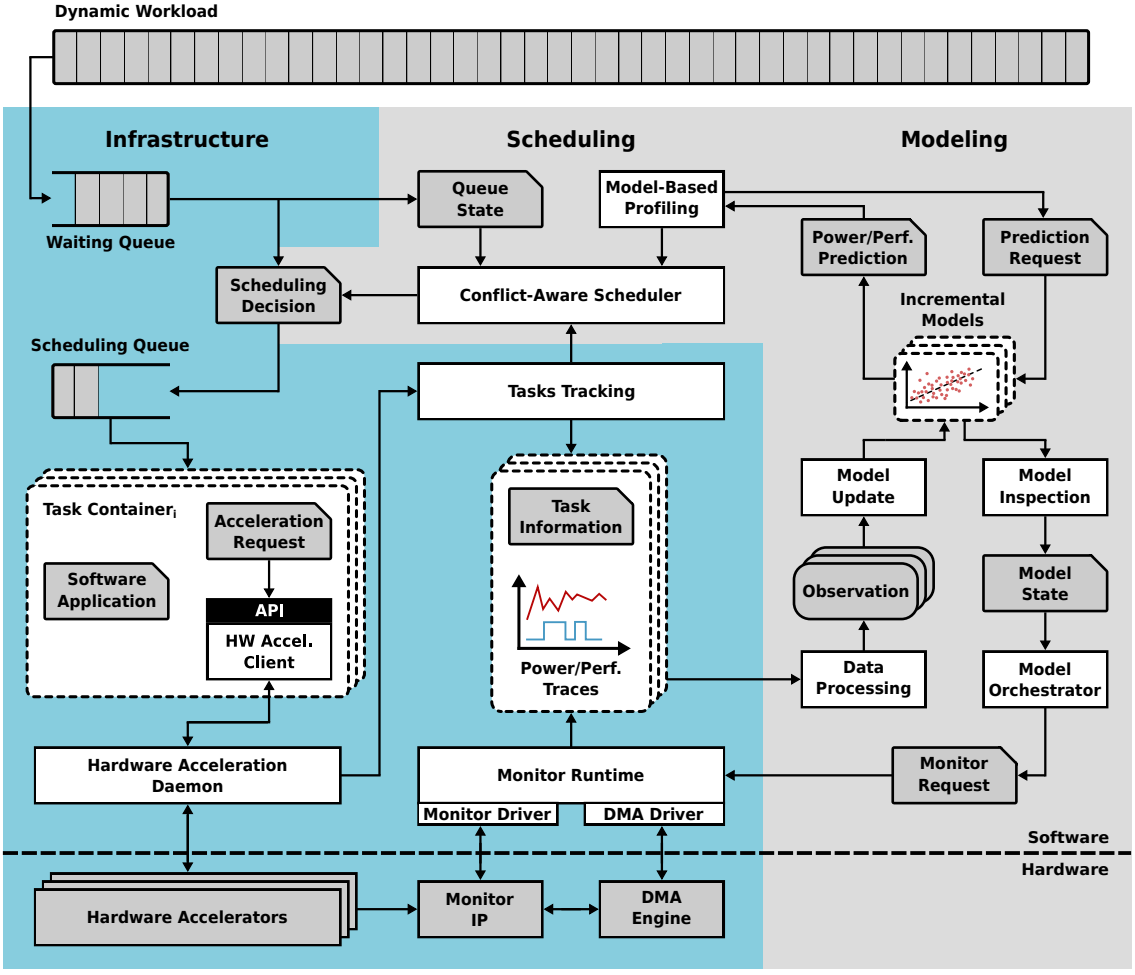


Figure 2-1: Detail of the infrastructure proposed in this Thesis.

In the context of this Thesis, a workload is a series of user tasks that arrive over time, each composed of a software section to be executed on the CPU of the device

and one or more hardware sections to be offloaded onto the FPGA fabric. The proposed infrastructure is based on a two-queue structure. Each of the workload tasks is placed in a waiting queue upon arrival and then moved to a scheduling queue for a subsequent execution, based on a specific scheduling policy and the available FPGA resources. The tasks placed in the scheduling queue, as shown in the block named task container of Figure 2-1, are executed on the device. The software section runs on the CPU, while the hardware sections are implemented as reconfigurable hardware accelerators in the fabric of the FPGA, on top of the ARTICo^3 infrastructure. Note that tasks are virtualized by encapsulating them in containers, enabling their implementation on any node of the cloud-edge continuum, making this deployment independent of the underlying hardware.[1] Moreover, ARTICo^3 has been extended to support a client-daemon architecture to manage simultaneous acceleration requests in a multi-tenancy fashion where multiple users share the reconfigurable fabric, optimizing the return on investment in the technology.

Tasks are tracked during their lifespan, extracting information such as arrival time, start and stop time, or reconfigurable slots used for their acceleration. Meanwhile, a monitor architecture implemented in hardware and controlled by a software application in the CPU acquires synchronized power consumption and performance traces from the hardware accelerators running on the FPGA at run time. Together, the collection of task information and hardware accelerator traces, managed by the proposed monitor framework, provides run-time power/performance trace acquisition of dynamic workloads in reconfigurable multi-accelerator systems.[2]

The infrastructure proposed in this section is available online and has been released under an open-source license.[3]

Original contribution 2-1 *An open-source platform-agnostic infrastructure to orchestrate, deploy, and monitor dynamic workloads on reconfigurable hardware in the cloud-edge continuum.*

[1] While containerization makes the deployment independent of the hardware underneath, the target nodes require ARTICo^3 support (more details in Section 2.3.1).

[2] The traces generated with the monitor framework are then leveraged in Chapter 3 to perform run-time characterization of dynamic workloads.

[3] https://github.com/des-cei/fpga-workload-manager/

2.2 State of the Art

The state-of-the-art work found in literature related to the management of resources in reconfigurable multi-accelerator systems is analyzed in this section. The section is divided into i) the architectures to perform hardware acceleration in reconfigurable multi-accelerator systems, ii) the support for FPGAs in the cloud-edge continuum, and iii) the monitoring of power consumption and performance in FPGAs.

2.2.1 Architectures for Reconfigurable Multi-Accelerator Systems

Deploying FPGAs in the cloud-edge continuum requires more than just the actual hardware. It demands a reconfiguration framework capable of abstracting hardware diversity, automating platform generation, and providing reliable run-time control in distributed and heterogeneous environments. Without such a framework, integrating FPGAs into cloud-edge continuum infrastructures becomes a complex process, limiting portability, adaptability, and overall system efficiency.

This section analyzes the state of the art in such frameworks, focusing on solutions that address the three essential layers:

- **Architecture:** defines the organization of static and dynamic FPGA regions, reconfiguration granularity, interconnections and DPR capabilities.
- **Design-time toolchain:** Automates platform creation, accelerator wrapping, integration into the DPR flow and bitstream generation.
- **Runtime management:** provides the software mechanisms to load, replace, and coordinate accelerators in the device.

From the wide range of related works, only those that integrate all three layers into a unified solution are considered. Approaches limited to a single or dual layer, such as architecture-only (e.g., DyRACT [Vipin'14], LEAP [Fleming'14]), toolchain-only (e.g., TaPaSCo [Heinz'21], TaPaSCo-AIE [Heinz'24]), or runtime-only (e.g., ZUCL 2.0 [Pham'19], RIFFA [Jacobsen'15]), while valuable in specific contexts, lack the completeness and portability required for an end-to-end cloud-edge FPGA deployment framework and are therefore excluded from the main comparison. The works that meet these criteria are presented below, describing each of the layers.

VAPRES

The Virtual Architecture for Partially Reconfigurable Embedded Systems (VAPRES) framework [Jara-Berrocal'10] divides the FPGA into a control region with a soft-core processor and a data processing region formed by Reconfigurable Streaming Blocks (RSBs) connected through a custom switch network. It supports DPR for loading accelerators and state save/restore for seamless hardware switching. Its companion toolchain, the VAPRES SoC Builder, generates the base system from user-defined parameters and integrates accelerators written in HDL or HLS, offering templates for simplified HLS development but requiring pre-partitioned applications. At run time, a Linux-based manager handles deployment requests by allocating resources, placing modules, reusing bitstreams when possible, and configuring interconnections to build complete stream-processing systems on demand.

FUSE

The Front-end USEr (FUSE) framework [Ismail'11] connects hardware accelerators to a host CPU over a bus and exposes them as OS-visible hardware threads through a two-layer abstraction that hides hardware details from software. While DPR could be supported, it is not evaluated in their presented work. Lacking a dedicated automated toolchain, accelerators must be manually developed and wrapped for their execution as a hardware thread. Its runtime extends the Linux scheduler to transparently offload Portable Operating System Interface (POSIX) threads to hardware when available, using shared memory for communication and device drivers to access control/status registers that manage the state of the hardware threads.

ReconOS

ReconOS [Agne'14] presents accelerators as OS threads connected to the CPU via a bus and extended with hardware synchronization primitives to ensure safe interaction with software threads. Its template-based toolchain generates hardware/software projects for HDL or HLS accelerators but requires manual partial reconfiguration floorplanning. At run time, delegate threads handle communication between the OS and hardware threads, providing full POSIX compatibility to ease the portability of multithreaded software.

ARTICo^3

ARTICo^3 [Rodríguez'18] implements a regular array of uniform reconfigurable slots connected via a shared bus, with optional Dual Modular Redundancy (DMR)/Triple Modular Redundancy (TMR) reliability features (specific for safety-critical applications) that require no accelerator modifications. The uniform slot size and interface allow module relocation for flexible placement and run-time DPR management. Its automated toolchain builds the complete system, integrates accelerators, and ensures reproducible, bitstream-compatible deployments. The Linux-based runtime provides utilities for slot assignment, module relocation, and DPR control.

Xilinx Vitis + XRT

The Vitis framework defines vendor-optimized static and dynamic FPGA regions, tailored to AMD boards. It offers a mature toolchain supporting HLS, C/C++, and Open Computing Language (OpenCL), with automated hardware/software integration and DPR via xclbin containers that encapsulate bitstreams and metadata. At run time, the Xilinx RunTime (XRT) library provides a unified API for kernel execution, DPR control, memory transfers, device queries, profiling, and multi-kernel execution management.

Intel + OpenCL

Intel's OpenCL-based approach uses vendor-defined static and dynamic partitioning, typically tuned for high-throughput kernel execution. The toolchain compiles OpenCL host and kernel code into bitstreams, supporting kernel replication and pipelining within a single flow. Run-time management is handled entirely through the OpenCL API, which supports kernel launch, memory operations, and FPGA reconfiguration.

Table 2-1 summarizes the main characteristics of the full-stack FPGA reconfiguration frameworks analyzed in this section. The comparison covers their architecture, toolchain, runtime APIs, and their capabilities regarding DPR and module relocation.

From the comparison, it is clear that each framework offers valuable strengths for different contexts. VAPRES delivers streaming efficiency and a well-structured separation between control and processing regions; however, its fixed pipeline structure limits adaptability to more irregular or control-intensive workloads. FUSE and ReconOS integrate hardware acceleration into familiar OS-level abstractions, lowering the barrier for adoption, but provide limited flexibility in partial reconfiguration and lack scalable relocation capabilities. Vendor-specific solutions such as Vitis/XRT and Intel's OpenCL flow offer mature toolchains, broad language support, and robust runtime APIs, but they are tightly coupled to proprietary architectures, which constrains portability.

Within this scenario, ARTICo^3 has been the selected candidate as it offers a balanced compromise between portability and run-time flexibility. Its uniform slot

Table 2-1: Comparison of full-stack FPGA reconfiguration frameworks.

Framework	Architecture	Toolchain	API	DPR	Reloc.
VAPRES [Jara-Berrocal'10]	CPU + pipeline	HDL/HLS	C	Yes	No
FUSE [Ismail'11]	HW threads	Manual	POSIX	Basic	No
ReconOS [Agne'14]	HW/SW threads	Template	POSIX	Yes	No
ARTICo^3 [Rodríguez'18]	Uniform slots	Template	C/C++	Yes	Yes
Vitis + XRT	Vendor platform	Full (hw/sw)	C/C++/OpenCL	Yes	No
Intel OpenCL	Vendor platform	Full (OpenCL)	OpenCL	Yes	No

AND (+) | **OR** (/)

Toolchain
Degree of automation in FPGA build flow and integration

Reloc.
Support for module relocation at run time

architecture, relocation support, and built-in reliability mechanisms make it well-suited to heterogeneous and distributed environments, while the automated toolchain ensures reproducibility and consistent integration. Nonetheless, to fully address the demands of cloud-edge deployments, ARTICo^3 would benefit from certain extensions, such as cloud-grade FPGAs support and native multi-tenancy. These additions will be implemented as part of this Thesis in Section 2.3.1.

2.2.2 Architectures to Support Cloud-Edge Continuum on FPGAs

Over the past few years, most works targeting hardware acceleration with FPGAs were addressing either cloud computing [Leeser'21] or edge computing [Xu'22]. A noteworthy aspect of such works is that computing resources are typically treated as monolithic entities tailored explicitly to the task at hand. However, as stated in [Gkonis'23], the cloud-edge continuum requires moving away from ad-hoc implementations toward flexible and scalable computing architectures, enabling their effective implementation on the diverse set of computing devices present in the continuum. Additionally, hardware-accelerated user applications must be portable, ensuring seamless deployment across all continuum nodes.

The concept of microservices proves helpful in addressing these needs, as it enables computing architectures with higher flexibility and scalability [Kamisetty'23]. When implemented in the field of FPGAs, the idea essentially involves allowing users to request the FPGA fabric as a microservice that provides hardware acceleration capabilities. With proper implementation, this approach can effectively virtualize the underlying hardware, even enabling simultaneous access to multiple tenants. Users simply request acceleration services, while the architecture handles all implementation details behind the scenes.

Nevertheless, the studies found in the literature that utilize microservices for hardware acceleration in FPGAs are still primarily focused on supporting a single specific computing resource. Specifically, most of the state-of-the-art works focus on

high-end FPGAs that are typically employed in the cloud domain by attaching them to a host CPU via a Peripheral Component Interconnect Express (PCIe) connection. An example of a microservice architecture for PCIe-based FPGA is presented in [Ojika'19]. The architecture requires two nodes, a server and a worker node, connected via a high-speed network. A pool of predefined kernels is made available for hardware acceleration in the device. Users can concurrently query the execution of any of them through an API implemented as a web application, which triggers the deployment of a container in the reconfigurable node. Each container then communicates with the working node via a web socket, indicating the appropriate kernel to be accelerated. The worker node, also encapsulated in a container, performs the actual hardware acceleration in the FPGA using dedicated OS drivers. The use of containers grants portability to other devices; however, the worker container is in privileged mode, which grants it most of the rights of the host (e.g., device access and relaxed security policies), thereby bypassing its intended isolation.

The authors of [Long'20] present a slightly different approach, again for PCIe-based FPGAs. Once more, user applications are deployed as containers in the host CPU of the FPGA, but in this case, the management of these containers and the user interaction with them are handled through the Kubernetes orchestration tool. In contrast to the previous work, users must include hardware-specific instructions in their applications to manage directly the access to the FPGA hardware, which prevents multi-tenant execution and also requires using the privileged container mode. Similarly, in [Bacis'20], the authors propose a hardware abstraction layer based on OpenCL. User applications can exploit OpenCL primitives to query hardware accesses and data transfers. A custom OpenCL library (one per user) receives those queries and transfers them to a device manager module (one per FPGA instance), which performs the corresponding hardware accesses transparently to the user. Decoupling user applications from the FPGA enables multiple users to submit acceleration requests to the same FPGA.

These approaches are restricted to PCIe-based FPGAs, which are generally intended for high-end cloud-related scenarios. This is a limitation for the cloud-edge continuum, where lower-end edge devices must also be supported. However, there is limited literature about the use of microservices on edge FPGA systems. For instance, authors in [Lallet'18] present a framework to manage hardware acceleration in lower-end Multiprocessor System-on-Chip (MPSoC) FPGAs, even though it still requires a PCIe connection for efficient data transfers from user applications to the FPGA fabric. The concept here is similar to that of previous works; user applications are deployed as containers on a server node. This server node performs the hardware acceleration in an FPGA that is attached using a PCIe connection. Its differential aspect is that the control and data movement processes are performed using distinct approaches. For managing the FPGA reconfiguration and kernel execution, the application container communicates via an Ethernet port from the server node to a dedicated processor in the FPGA. This dedicated processor is responsible for low-level hardware access and can handle multiple user queries. In turn, the data from user applications is

transferred from the containers on the server node to the FPGA fabric via the PCIe port. For this purpose, authors employ Riffa [Jacobsen'15], a framework that, among other purposes, performs efficient PCIe-based data transfers. An example specifically designed for edge devices is presented in [Al Qassem'20]. A web application running on a host CPU is accessible to the users, who can deploy hardware acceleration in the FPGA by submitting the configuration file (i.e., bitstream) through the web application. The reconfiguration and execution processes of the kernels are handled by the host CPU via a dedicated FPGA port, one query at a time. Every architecture component is encapsulated in a container to achieve portability.

In contrast with state-of-the-art works presented in this section, the work of this Thesis is designed to be compatible with FPGAs from cloud to edge, whether they feature a PCIe connection, on-chip processors, or a soft-core processor placed in the FPGA fabric. As a hardware-abstraction layer, the microservice architecture implemented is conceived as a daemon-client structure that effectively hides low-level FPGA operations (e.g., reconfiguration) from the user, as well as enables multi-tenant execution. User applications, which are encapsulated in containers, communicate with the daemon using an API that is platform-agnostic, enhancing portability. Table 2-2 presents a qualitative comparison between the state-of-the-art works reviewed in this section and this Thesis proposal.

Table 2-2: State of the Art - Support of the cloud-edge continuum on FPGAs.

Architecture	Platform	User Abstraction	Portability	Multitenancy
[Ojika'19]	PCIe	Control + Kernels	Arch	✓
[Long'20]	PCIe	Control	Arch	✗
[Bacis'20]	PCIe	Control + Kernels	✗	✓
[Lallet'18]	PCIe + OCP	✗	Kernels	✓
[Al Qassem'20]	PCIe	Control	Arch	✗
This work	PCIe/OCP/SCP	Control + Kernels	Arch + Kernels	✓

AND (+) | **OR** (/)

Platform
PCIe: PCIe-connected host required
OCP: on-chip processor required
SCP: soft-core processor required

User Abstraction
Control: low-level hardware details about the control of the architecture
Kernels: low-level hardware details about the kernel descriptions

Portability
Arch: architecture components
Kernels: user kernel descriptions

2.2.3 Power and Performance Monitoring on SoPC

The primary purpose of the SoPC monitoring solutions found in literature is to make observable the components of the processing system (e.g. CPU, memory) [Aldham'11, Matthews'10, Nadimpalli'16, Patrigeon'18, Vaishnav'18]. In contrast, monitoring programmable logic components (e.g., hardware accelerators) still requires exploration.

The leading FPGA vendors provide specific commercial monitoring solutions as debugging tools for their platforms. AMD, for instance, offers the Integrated Logic Analyzer (ILA) [Xilinx'21], a tool embedded in the AMD toolchain that instruments the hardware configured in the FPGA fabric, enabling developers to observe the transition of internal signals and analyze the system's behavior. Likewise, Intel provides the Signal TAP [Intel'24] logic analyzer, a similar alternative for their platforms. These solutions rely on a host machine running proprietary software to monitor the platform and require significant resources, restricting them to design-time debugging and verification purposes. Another monitoring solution is the AMD Xmultil [Xilinx'23b], designed explicitly for the KRIA System on Module (SoM).[4] This utility provides a library for run-time monitoring of non-functional metrics, such as temperature and power consumption, across the entire SoM. However, it is specific for KRIA SoM platforms, it does not allow any customization regarding the features to be monitored, and presents a limited sampling rate (1 sample/s).

On the academic side, the work in [Goeders'17] introduces an HLS tool that automates the inclusion of monitors in the RTLs code generated by the tool, for debugging. This approach has been validated on the LegUp HLS platform [Canis'13], where the actual hardware and the monitors are inferred from the descriptions in *C++*. Similarly, the authors in [Hammouda'17] propose an HLS tool that generates systems together with run-time verification monitors, extracted from user-defined assertions in the *C* description of the system. On a similar note, for debugging/verification purposes, it is worth mentioning the work in [Lee'15]. The authors propose a system-level observation framework that uses customizable observation probes to perform run-time analysis and verification of software and hardware components.

Although these approaches are tailored for debugging and verification purposes, some works in the literature also focus on system performance monitoring. This is the case of the work presented in [Fanni'19], where the Performance API (PAPI)[5] is extended to read not only the Performance Monitoring Counters (PMCs) present in CPUs, but also application-specific PMCs embedded in the FPGA fabric, targeting coarse-grained reconfigurable systems. A similar approach is followed in [Valente'21], where the authors introduce system sniffers to gather monitoring information from both software and hardware components, making them observable.

In contrast, power monitoring is the primary focus in the works presented in [Najem'17] and [Zoni'18]. Both solutions use post-place and route simulations of

[4] An SoM is a compact computing module that integrates an SoC, memory, power management, and common inputs/outputs on a small board that plugs into a base board.

[5] https://icl.utk.edu/papi/

RTL descriptions to obtain power consumption estimates. That data is then fed to ML models to train them for run-time predictions of power consumption. A different approach is followed in [Akgün'20], exploiting an Inter-Integrated Circuit (I²C)-based Power Management Bus (PMBus) communication protocol to gather voltage and current, and compute power consumption. This approach is then used to implement dynamic voltage and frequency scaling techniques.

Contrary to the works analyzed in this section, the monitoring solution proposed in this Thesis is designed to simultaneously address the three discussed monitoring needs (i.e., debugging and verification, performance, and power consumption) in FPGA-based systems. It can register changes in internal hardware signals of the FPGA fabric, similarly to commercial solutions, but is lightweight and does not require proprietary software or an additional host machine. Moreover, due to this internal signal registration, the proposed solution can monitor the execution of hardware accelerators to extract performance metrics (i.e., from start/ready signals or communication commands). Regarding monitoring power consumption, this proposal performs real, rather than simulation-based, power consumption measurements from a set of sources (i.e., on-chip Analog-to-Digital Converters (ADCs) or external power measurement boards). The approach is similar to the work presented in [Akgün'20], but offers significant improvements in sampling frequency. A qualitative comparison between the works found in the literature and the proposal of this Thesis is presented in Table 2-3.

Table 2-3: State of the Art - Power and performance monitoring on FPGAs.

Monitor	Debug and Verification	Performance	Power Consumption
[Goeders'17]	RTL instrumentation	—	—
[Hammouda'17]	RTL instrumentation	—	—
[Lee'15]	RTL instrumentation	—	—
[Fanni'19]	—	PMCs	—
[Valente'21]	RTL instrumentation	PMCs	—
[Najem'17]	—	—	Simulation-based
[Zoni'18]	—	—	Simulation-based
[Akgün'20]	—	—	Real measurements (I²C)
This work	HW signal probes	HW signal probes	Real measurements (SPI)

2.3 Node-Level Infrastructure

The main components of the proposed infrastructure, from the perspective of a single node, are described in this section. First, the extension introduced to the ARTICo^3 framework is presented, required to satisfy the hardware acceleration needs of the cloud-edge continuum. A virtualization mechanism is presented to enable the portability of tasks between nodes. Then, the monitoring framework developed to gather run-time power consumption and performance is described. And finally, these components are holistically combined to form a workload management infrastructure.

2.3.1 ARTICo³ Extension

The ARTICo^3 framework was originally designed to implement hardware acceleration in MPSoC FPGAs, which integrate a host processor and reconfigurable logic on the same device. This approach was convenient at the moment, since it is an FPGA format commonly used in industrial applications. However, due to the adoption of FPGAs as cloud computing devices, where the FPGA is connected to a host CPU via PCIe, vendors now ship them as reconfigurable-logic-only cards as well. Therefore, an extension of the ARTICo^3 framework is required to support the PCIe-based FPGAs present in the cloud layer of the cloud-edge continuum.

Furthermore, cloud-grade FPGAs are equipped with a large amount of reconfigurable logic. This feature, combined with run-time changes in part of the logic due to their DPR capabilities, enables sharing the same hardware device between multiple users. This capability enables the implementation of multi-tenant execution, which optimizes the return on investment in the technology by maximizing resource utilization. However, the original ARTICo^3 implementation does not support this model of computation, which also requires an extension of the framework.

ARTICo^3 for Cloud-based FPGAs

The ARTICo^3 framework communicates the host processor with the reconfigurable logic using memory-mapped registers and DMA transfers over on-chip communication buses, since MPSoC FPGAs present them on the same device. However, cloud FPGAs rely on a high-speed PCIe connection. Therefore, to enable the use of ARTICo^3 in cloud FPGAs, its original implementation has to be adapted to the PCIe interface.

Specifically, three main aspects of the ARTICo^3 architecture must be adapted to support PCIe: i) access to components within the reconfigurable logic of the FPGA, ii) data transfers between the host processor and the reconfigurable logic, and iii) the dynamic reconfiguration process. To enable these functionalities over a PCIe connection, device manufacturers provide IPs such as the XDMA [Xilinx'24], which serves as a bridge between the PCIe interface and the internal communication buses of

the reconfigurable logic. The implementation of these ARTICo^3 adaptations, detailed below, is based on the use of the XDMA IP. Additionally, Figure 2-2 illustrates a cloud FPGA architecture with the adapted ARTICo^3 components integrated.

- **Access to the reconfigurable logic and efficient data transfers:** in the original ARTICo^3 implementation, the communication from the host CPU to the reconfigurable logic is performed via standard Advance eXtensible Interface (AXI)-lite interfaces that are register-based. In turn, an AXI-full interface (a standardized interconnection capable of performing burst-based transactions) is used together with a DMA engine to efficiently transfer data between the host CPU and the reconfigurable logic. This approach enabled i) the configuration and control of hardware accelerators and other components in the reconfigurable logic as memory-mapped peripherals, and ii) the movement of data through the system. To adapt to the PCIe connection, the XDMA IP has been included as part of the ARTICo^3 architecture. Through this component, the host CPU can access the memory-mapped components of the reconfigurable logic, and it can perform efficient DMA transactions (as the XDMA integrates a DMA engine). In addition to this modification in the ARTICo^3 architecture, the companion runtime library has been adapted to use the corresponding XDMA Linux driver instead of the one originally used. Note that the ARTICo^3 toolchain used to generate the software and hardware components has also been adapted.

- **Reconfiguration process:** the DPR process implemented originally in ARTICo^3 relies on the Processor Configuration Access Port (PCAP) [Xilinx'23e], which enables DMA-powered logic reconfiguration. However, the PCAP is specific to devices that feature a processor embedded in the FPGA, which is not the case in PCIe-based devices. An alternative when the PCAP is not present is to rely on the Internal Configuration Access Port (ICAP), a configuration port accessible from reconfigurable logic. For managing reconfiguration through the ICAP [Xilinx'25a] a convenient option is to use the High Bandwidth ICAP (HBICAP) IP [Xilinx'19a], which essentially hides ICAP specifics and enables the user to send the configuration bitstream via a DMA transfer. Therefore, the approach followed is to send the bitstream to the HBICAP as a DMA transfer leveraging the XDMA as in the previous case. From a software perspective, the reconfiguration flow of the original ARTICo^3 framework, which used a dedicated Linux driver to handle reconfiguration through the PCAP, has been replaced with this procedure using the XDMA driver to perform bitstream configuration through HBICAP.

The adaptations proposed in this section enable hardware acceleration implementation on PCIe-based FPGAs. These adaptations have been integrated into the original architecture, the runtime library, and the toolchain of ARTICo^3. Now, ARTICo^3 supports all types of platforms found in the cloud-edge continuum.

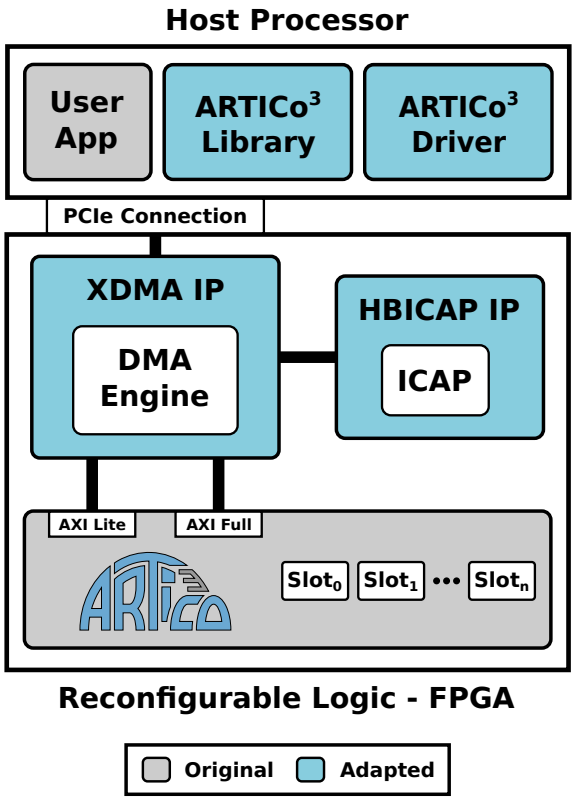


Figure 2-2: Block diagram of a cloud-oriented FPGA integrating the ARTICo^3 components.

Multi-Tenant Support on ARTICo^3

Since the ARTICo^3 framework is intended to be used in MPSoC FPGAs equipped with limited resources, user applications can access the complete reconfigurable logic without resource-sharing mechanisms. However, the large devices tailored for cloud computing could benefit from multi-tenant execution by sharing the logic among multiple users, which requires the extension of the ARTICo^3 framework.

From a hardware perspective, the ARTICo^3 framework does not require a change. In contrast, the runtime library requires a major redesign. The proposal is to move to a user/daemon execution model. Originally, the runtime library was a passive collection of functions that got called on demand by the user application. However, with the redesign, the runtime library runs as a background daemon, supporting asynchronous event-driven operations and enabling multi-user workflows. User applications submit asynchronous requests, which the daemon handles. Conceptually, the library is being split into two distinct components with the following philosophy:

- **Daemon service:** implements all the operations required to perform hardware acceleration in the FPGA (i.e., logic reconfiguration, acceleration execution and data movement). It works as a background service on the device. When a user queries an operation, its completion is delegated to a worker thread, allowing multiple requests to be handled simultaneously. Serving user queries is handled

with sockets, while management operations that cannot be performed in parallel with others (i.e., logic reconfiguration and DMA operations) are performed using mutex synchronization primitives.

- **User runtime:** serves as an interface between the user applications and the daemon service. This runtime exposes an API to the user, which contains all the operations required for hardware acceleration. Each method this library exposes then queries the operation to the daemon service through the mentioned communication mechanisms. Note that multiple users can utilize this library simultaneously, as the daemon performs the actual hardware acceleration.

A representation of this user/daemon execution model is shown in Figure 2-3, where it is compared with the original ARTICo^3 execution model. The figure illustrates a simplified example of the steps involved in accelerating a task in hardware. In the original approach, both the user code and the runtime library were embedded within the same application binary. This design had a major limitation: multiple user applications could potentially compete for access to the hardware, as there was no system-wide mechanism for coordination or resource management.

The execution model proposed in this Thesis addresses this limitation by introducing a system-wide background service, referred to as the daemon service, that centralizes the management of hardware interactions. With this approach, user applications issue acceleration requests to the daemon, which delegates each request to a dedicated worker thread. Synchronization mechanisms within the daemon ensure that concurrent access to shared hardware resources is handled safely and efficiently. This execution model enables multi-tenant hardware acceleration in a controlled and coordinated manner, allowing the users to share the FPGA resources and maximize their utilization.

Original contribution 2-2 *The extension of the ARTICo^3 framework for cloud-based FPGAs and multi-tenant execution.*

2.3.2 Virtualization of Hardware Acceleration

Deploying a hardware acceleration task on any node, agnostic to the underlying hardware, is a must when working in a cloud-edge continuum scenario because user applications are requested to be deployed on the continuum, not on specific hardware.[6] Therefore, tasks must be portable between nodes without user intervention, which is known as virtualization. Achieving this goal requires both sides of the user application, the software (i.e., part running in the CPU) and hardware (i.e., reconfiguration, data movement to reconfigurable logic, the bitstream itself) components, to be portable between nodes. Fortunately, the new execution

[6] Note that a node will be eventually selected for the deployment, but transparently to the user.

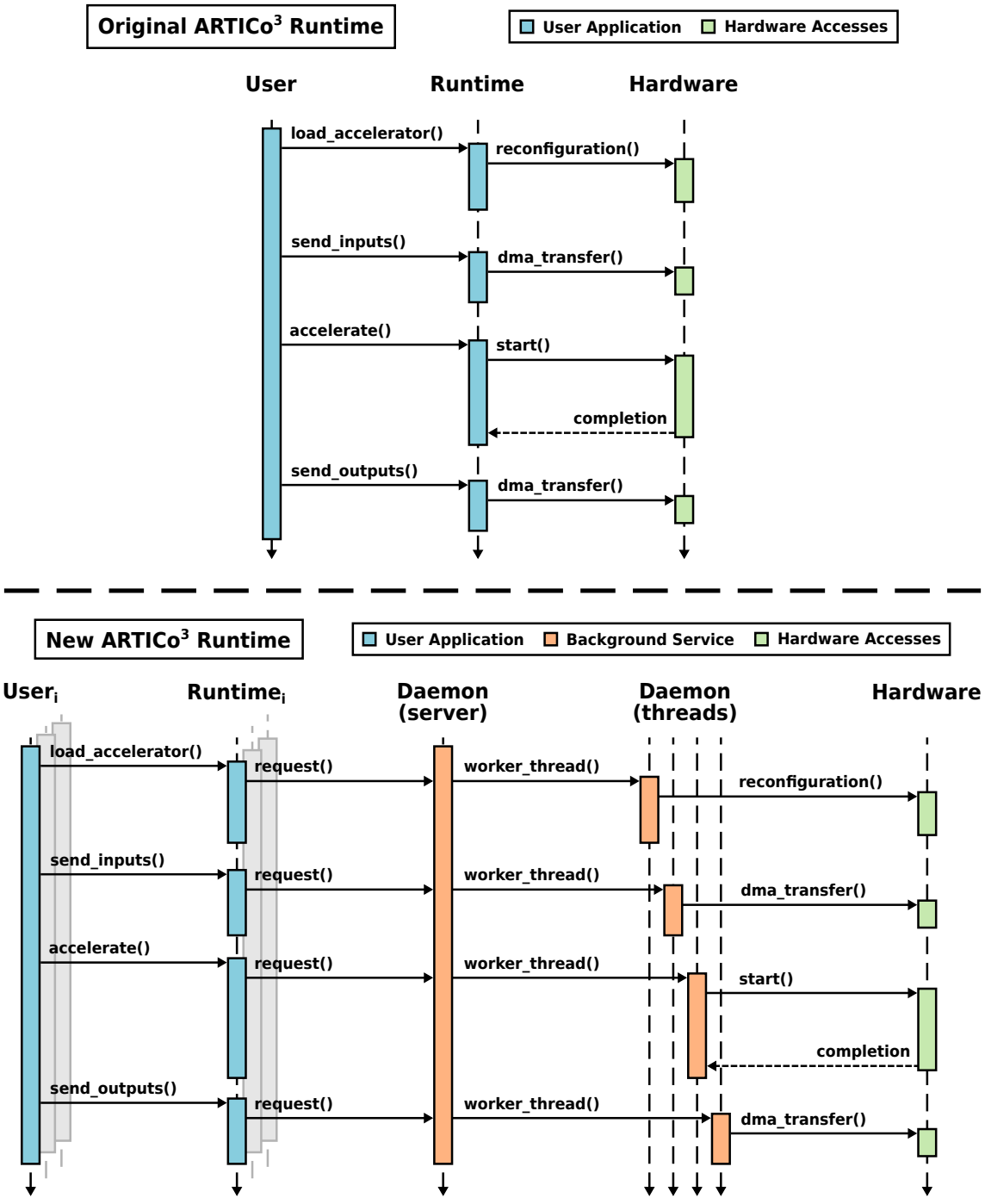


Figure 2-3: Comparison between the new and original execution models of ARTICo^3. Note that each task (i.e., box) is represented by a color based on its execution domain: blue, part of the user application; orange, runs as a system-wide background service; and green, handles hardware interactions.

model implemented in the previous section (i.e., the user/daemon approach) solves part of the problem, as the management of the reconfigurable accelerators is now implemented locally on the device, separate from the user application, but as a standalone service.

To virtualize the software components, which refers to the user application and the mentioned runtime user library, containerization is the proposed solution [Bentaleb'22]. This solution is ideal because it isolates software applications, ensuring that every dependency is self-contained. This isolation allows for portability to other nodes with negligible overhead due to its lightweight implementation. Note that the container building process enables the creation of multi-architecture images, which contain multiple versions of certain components that require tailored versions for specific CPU architectures.

Figure 2-4 shows a representation of how the user application, embedded in a container, is executed on any desired node. Each of the nodes would have the appropriate version of the ARTICo^3 architecture implemented on hardware, as well as its corresponding daemon service versions running in the CPU. To run the user application, a container must be deployed to the target node, which queries the user commands to the daemon running on the node and performs hardware acceleration.

2.3.3 Monitoring Framework

The workload characterization methodology proposed in this manuscript, which will be described in Chapter 3, requires the acquisition of power consumption and performance metrics during the execution of the workload. Hence, a monitoring mechanism needs to be available on the system, as part of the infrastructure, for trace acquisition at run time. In fact, as analyzed in Section 2.2.3, monitoring has evolved from a debugging tool to a strategic enabler for run-time performance analysis, facilitating optimizations such as workload redistribution [Böhm'22] and operating condition adjustments [Fanni'18]. However, it has been observed that most widely extended monitoring solutions remain limited in scope, primarily focused on debugging functionalities (e.g., AMD ILA, Intel Signal Tap Logic Analyzer). These solutions lack cross-platform support, synchronized power-performance measurements, and detailed execution trace capturing, creating a gap in monitoring capabilities for FPGA SoC.

To address these gaps, a modular and composable power-performance monitoring framework tailored for FPGA SoC has been designed. The framework ensures cross-platform compatibility and real-time synchronization of power consumption and performance data.

The framework has been designed using a modular approach, as illustrated in Figure 2-5, to enforce flexibility and compatibility, which are primary requirements when dealing with the heterogeneous nature of systems featuring the cloud-edge continuum. A customization layer (located at the top-right side of the figure) enables

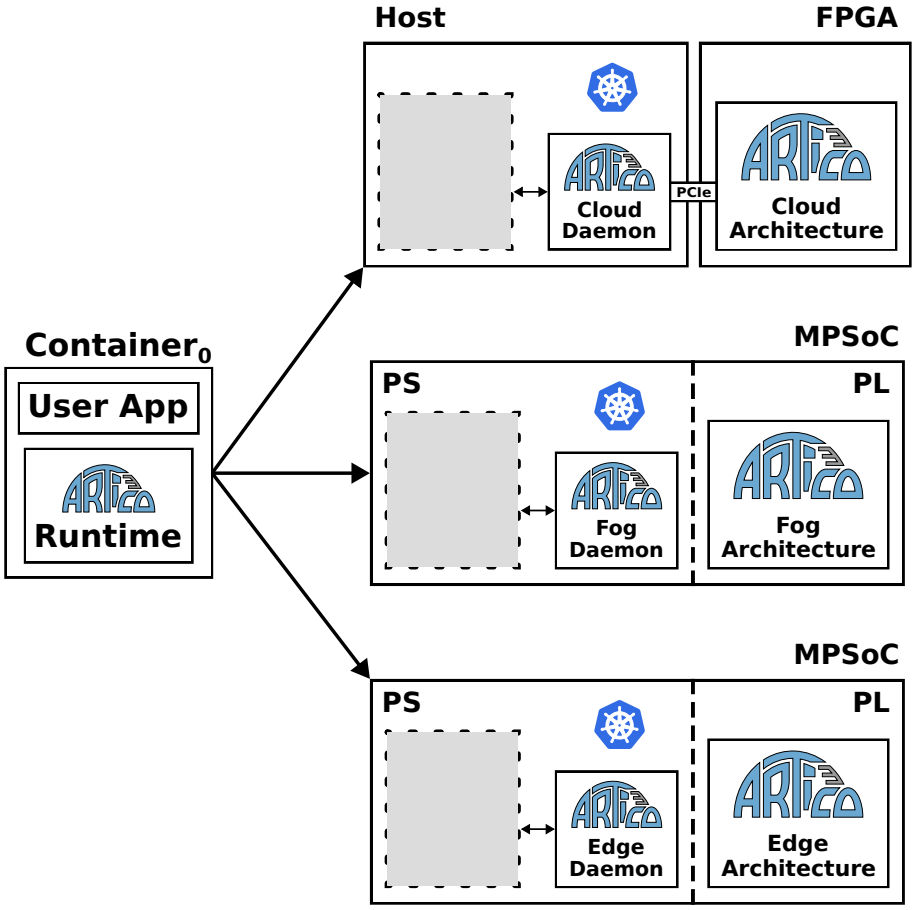


Figure 2-4: Representation of the virtualization of hardware accelerators with ARTICo[3].

users to adjust components, such as the power collection method, acceleration implementation, and device specification, according to their specific requirements. Each of these components, grouped into three categories, presents different flavors to accommodate the potential monitoring goals of users. These components are fully compatible, and any of them can be selected without modifying the system architecture. Moreover, the monitored traces can be generated in multiple formats depending on the goal. At the right of that figure, the representation of a generic monitoring infrastructure composed with this framework is depicted, which has also been represented in Figure 2-6 with a low-level block diagram. Each of the compounding layers of the monitoring framework is detailed below.

The monitoring framework proposed in this section is available online and has been released under an open-source license.[7]

[7]https://github.com/des-cei/fpga-monitor/

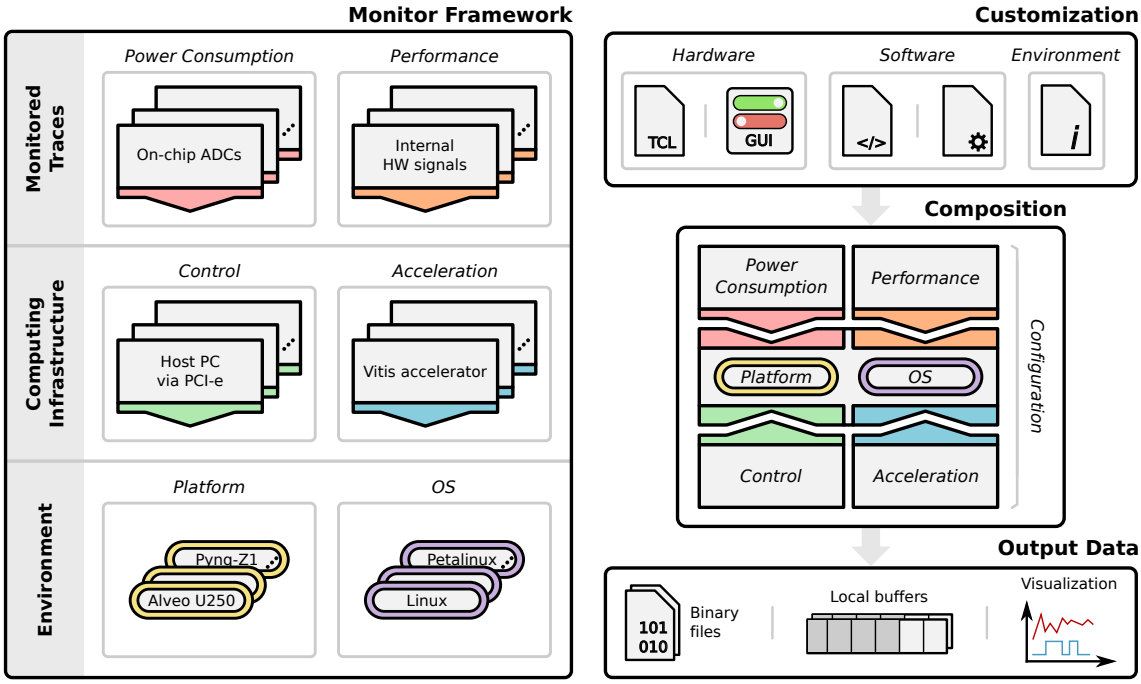


Figure 2-5: Overview of the monitoring framework.

Monitored Traces

The proposed framework enables power consumption and performance monitoring of the hardware accelerators implemented on the PL of an FPGA, together with the power consumption monitoring of the PS, where software runs. Indeed, the system could be generalized to other metrics such as temperature or memory access statistics. However, the focus is on these two since they are commonly considered the primary metrics for debugging, verification, or optimization purposes. Hardware and software components have been designed to address various application-dependent needs. The extraction process for each of these types of traces is explained next.

- **Power Consumption:** many commercial reconfigurable platforms are shipped with shunt resistors placed in dedicated power rails of the development boards, coupled with built-in on-board ADCs, making them accessible typically via an I²C bus [Xilinx'23d, Xilinx'25b]. This framework natively provides the required software stack to interact with several of these I²C-based ADCs, allowing the user to gather periodic timestamped power consumption measurements from the system and to store them in a local buffer in a transparent manner for the system designer. A configuration layer grants portability by enabling the user to set device-specific I²C communication settings.[8]

  However, this infrastructure is not available on all boards, and even those that have it, the sampling frequency limitation of the I²C interface makes it unsuitable

[8]Most of the I²C-based ADCs shipped in the platforms of the main manufacturers (e.g., AMD Xilinx) are already compatible, but the software stack can be extended to other devices.

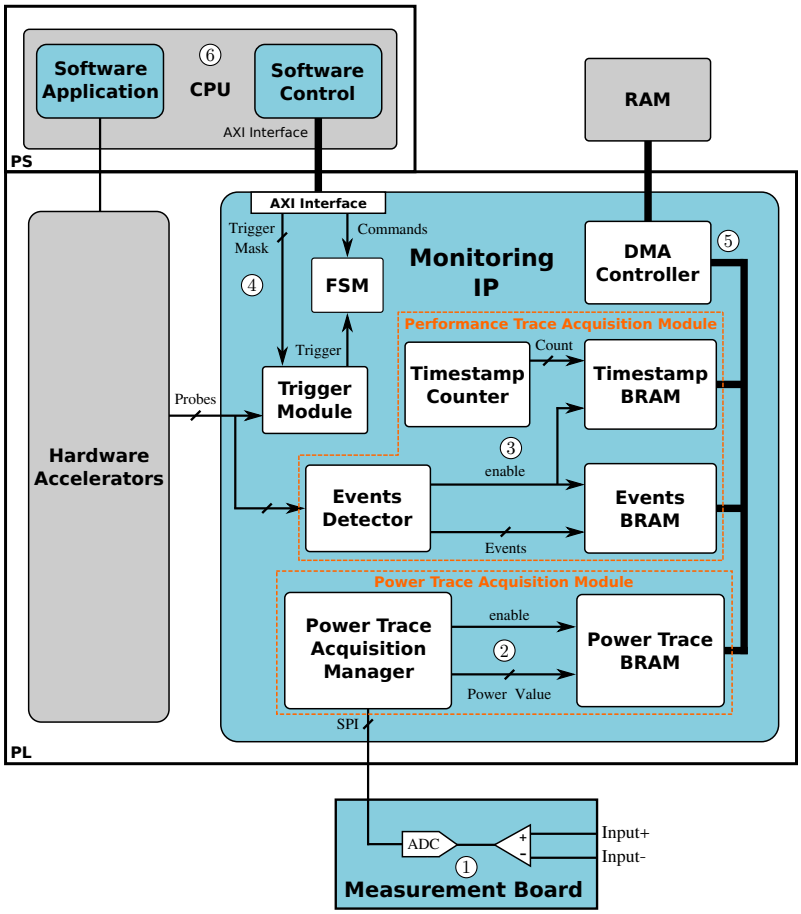


Figure 2-6: General overview of a monitoring infrastructure composed with the monitoring framework. Gray blocks represent components that are not part of the monitoring infrastructure, such as the hardware accelerators being monitored (on the left). Blue blocks represent monitoring infrastructure components, which consist of: i) power traces acquisition components (① and ②), ii) performance traces acquisition components (③), iii) infrastructure control components (④ and ⑤) and iv) software drivers (⑥).

for some applications. Hence, an external power measurement board with a Serial Peripheral Interface (SPI)-based ADC has been designed (① in Figure 2-6) and provided as a complementary measurement solution, enabling acquisition rates of up to 1 Msample/s (therefore, overcoming resolution limitations found in literature). The board features multiple channels that allow i) full-board power measurement suitable for USB-power devices, where the power supply is bypassed through the board and measured with a shunt resistor, and ii) measurement of dedicated power rails, either directly from the power rail using the shunt resistors present in the measurement board, or behind the shunt resistors already placed in the board to be monitored (if any). To minimize the overhead induced by the measuring process, the SPI controller that reads the ADC has been implemented in hardware on the FPGA fabric (② in Figure 2-6), allowing the communication process to occur in parallel with the rest of the

system operation. Apart from the SPI controller, the infrastructure includes the Block RAMs (BRAMs) to store the power consumption measurements on the FPGA and the required control logic.

Apart from these features, the framework is built to facilitate the integration of additional power consumption measuring methods and to remove power consumption monitoring when not needed.

- **Performance:** in FPGA-based systems, the hardware accelerators are generally controlled with a 1-bit start/ready signals (mapped to specific bits in a memory-mapped control register) or via communication bus exchanges that trigger execution and indicate completion. This happens in typical bus-based systems, for instance, AXI Stream, where ready/valid handshake signals are used.

  User-defined internal signal events are registered with their corresponding timestamps to monitor performance. This is achieved with specific hardware components (e.g., edge detectors or counters, typically) placed in the FPGA fabric (③ in Figure 2-6). When 1-bit signals are to be measured, each user-defined signal event is stored. If kernel execution is triggered by communication bus commands, 1-bit start/ready signals are crafted when start/stop commands are detected passing through the bus, unifying subsequent data processing.

Both power consumption and performance traces are sampled in parallel and stored with timestamps, enabling synchronization of both measurements.

Computing Infrastructure

The interaction between control, monitoring, and hardware acceleration in heterogeneous systems can vary significantly depending on the underlying computing platform. This framework enables users to compose a monitoring infrastructure that operates under any desired control method (e.g., hard-core processor) while monitoring different hardware accelerators regardless of the chosen implementation approach (e.g., AMD Vitis or ARTICo^3). The proposed framework is compatible with multiple frameworks, as explained below.

- **Platform Control:** the monitoring components are packaged as IPs that can be instantiated in the FPGA fabric as a memory-mapped peripheral, controlled from the CPU through a set of control registers. Apart from the hardware described in the previous section, the monitoring infrastructure contains a Finite State Machine (FSM) that orchestrates the synchronized trace acquisition processes already described. The control registers can be accessed to trigger the FSM to start the acquisition process and check its status. The IP features AXI ports to i) manage the infrastructure from those registers and ii) read the traces using a DMA controller (④ and ⑤ in Figure 2-6).

It is assumed that a microprocessor is always available in the target reconfigurable platform to control the infrastructure, as is the case with SoPC, which is the main target of this Thesis. A soft processor can be used instead if this is not the case. In the case of AMD Xilinx SoCs, the PS presents a hard-core processor and the PL contains the FPGA fabric. With this approach, the hardware components placed in the PL can be managed from the PS. As already mentioned, another possibility is connecting an FPGA device to a host CPU via a PCIe connection.

This monitoring framework is compatible with all these approaches and includes the required drivers and libraries to facilitate the management of the composed monitoring infrastructures.

- **Acceleration Infrastructure:** this framework is designed to be agnostic of the hardware acceleration implemented in the reconfigurable component of the system, even though this Thesis uses ARTICo^3 for hardware acceleration. As long as the hardware accelerators placed in the FPGA fabric expose either start/ready signals or AXI buses for controlling them, the resulting monitoring infrastructures composed with this framework will be compatible. The application domain is not a limiting factor either. This compatibility support has been tested with MDC [Sau'21], ARTICo^3 [Rodríguez'18] and AMD Vitis [Xilinx'23c], three relevant frameworks for building reconfigurable multi-accelerator systems already described.

Environment

Heterogeneous reconfigurable systems are offered with very different architectures and components, featuring numerous commercial solutions that integrate combinations of processors, reconfigurable hardware, and interconnections, commonly referred to as platforms. Typically, they are coupled with an entire portfolio of OS's to obtain a system suitable for the target application. Hence, a monitoring framework must support various platforms and OS's to meet specific user needs.

- **Platform:** a broad landscape of reconfigurable platforms is available in the cloud-edge continuum to meet the diverse requirements of different users. From SoC platform integrating an FPGA with a microprocessor on the same device to PCIe-based platforms. From low-end platforms with limited resources for edge computing to high-end platforms with extensive resources for cloud computing.

  To cover such a broad spectrum of platforms, the hardware part of the monitoring infrastructures provided by this framework is an HDL code to be packed as an IP managed using an AXI interface. Hence, the monitoring framework supports most available commercial platforms, just assuming that they are extensible via AXI peripherals.

- **Operating System:** the hard- or soft-core processor present in the system can be used either in bare-metal or with Linux, based on the user's needs. The framework includes the required software components (i.e., libraries, drivers) to allow the user to interact with any monitoring infrastructure composed with the framework from a user application (⑥ in Figure 2-6), leveraging a simple API to manage the infrastructure.

  Users can compile libraries and drivers according to their chosen microprocessor architecture and/or Linux kernel version.

Extraction of the Measurements

Traces monitored with this framework can be i) stored as binary files to be processed later, ii) placed in local buffers for further run-time operations, or iii) plotted as graphical representations using the visualization tool also included as part of the monitoring framework.

An example of the traces that can be obtained with this monitoring framework, as represented by the visualization tool, is presented in Figure 2-7. Notice these traces illustrate a combination of analog signals, PS power consumption (top) and PL power consumption (middle), alongside digital waveforms (start/ready signal pairs at the bottom) that reflect the activity of the reconfigurable accelerators. In this example, the application utilizes three functionally identical hardware accelerators, with signals 0, 1, and 2 representing the start/ready pairs for the three accelerators.

Monitor Customization

As already described, the proposed monitoring framework enables the composition of specific monitoring infrastructures tailored to particular use cases by combining any available option in each framework component. The component options that have been tested in advance are presented in Table 2-4. However, as stated before, many other variations could be included with minimal to no porting effort.

Table 2-4: Configuration space of the monitoring framework (only tested components).

Platform	OS	Power	Performance	Control	Acceleration
Pynq-Z1 Kria KV260 SoM Alveo U250 Zynq US+ ZCU102 Genesys 2	Yocto-Linux Custom Linux Petalinux Bare-metal	External board On-chip ADCs Platform-specific	Internal HW signals AXI bus	PS PL PCIe	ARTICo^3 MDC AMD Vitis

As depicted in Figure 2-5, the configuration stage gathers hardware and software parameters that define the desired monitoring infrastructure (e.g., from Tool Command Language (TCL) scripts and configuration files), along with environmental information such as platform and OS specifications. The framework leverages that

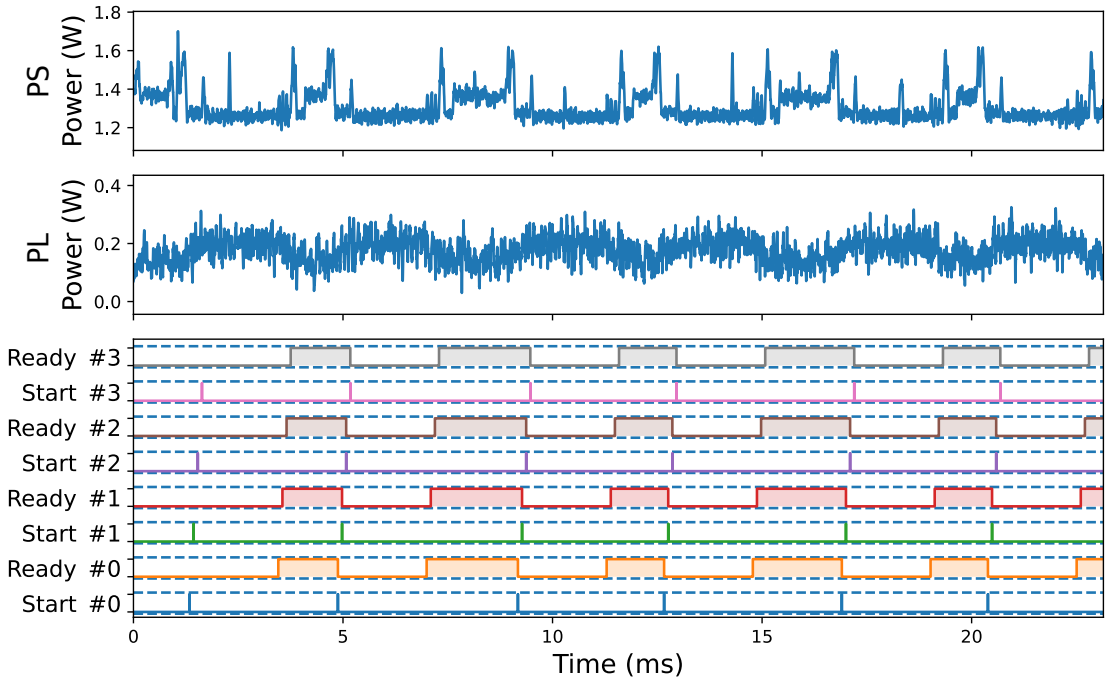


Figure 2-7: Power and performance traces obtained with the monitoring framework.

data to select the appropriate components to compose the requested monitoring infrastructure, as shown at the bottom of Figure 2-5. This process generates the user's hardware description (i.e., bitstream file) and software components (i.e., applications, libraries, drivers).

The modular approach of the proposed monitoring framework enables the integration of additional components beyond those presented in this section. The following are the main steps to configure the monitoring framework, which may include additional custom components not described here.

- **Hardware design:** any platform that provides an FPGA and a general-purpose core can be integrated. However, the provided automation is higher for the AMD platforms. A similar implementation effort could be done for other platforms in the future. The use of a Linux OS to manage the platform is recommended, making the integration smoother. On-board ADCs can be leveraged to measure power consumption when available. Otherwise, the proposed external board can be connected to serve this purpose. Finally, the target hardware accelerator(s) must be instantiated in the FPGA.

- **Monitoring IP configuration and connection:** the monitoring IP can be instantiated as a block in the Xilinx Vivado Block design environment and configured. The user can decide about the use of the external monitoring board, the number of signals to be monitored, and the amount of BRAMs allocated for storing traces. Then, the IP has to be connected to either the start/ready

signals coming from the hardware accelerator(s) or its communication bus for performance evaluation and synchronization. The identification of these signals is left to the user, as they are inherently dependent on the hardware accelerator architecture and interface.

- **APIs and execution:** a Linux OS running on the microprocessor can be used. The drivers for the monitoring IP would have to be inserted as modules in the kernel (via standard modprobe system call) and the device tree updated to account for this IP. These steps are platform-independent under Linux, and scripts for that are provided. The drivers are available as a C library to be compiled for bare-metal projects. In both cases, a standard API is called to start/stop monitoring.

In summary, this modular and composable framework provides a solution for monitoring power consumption and performance in reconfigurable FPGA-based systems. Its architecture leverages platform-agnostic interfaces to seamlessly integrate target-specific components (e.g., acceleration frameworks, ADCs, CPUs cores, and FPGA boards), enabling flexible plug-and-play monitoring and tracing capabilities across diverse user-defined scenarios. Moreover, it is available online as an open-source monitoring framework that could be used to provide monitoring capabilities to other projects beyond the scope of this Thesis.[9]

Original contribution 2-3 *An open-source, composable monitoring framework for synchronized power consumption and performance trace acquisition on reconfigurable multi-accelerator systems.*

2.3.4 Workload Management

This section combines the components described throughout this chapter to build a cohesive infrastructure that can efficiently manage the dynamic workloads faced by FPGA devices in the cloud-edge continuum scenario. This infrastructure is also capable of capturing run-time power consumption and performance traces. The proposed infrastructure features two distinct processes that control these procedures: the workload offloading process and the workload registration process.

Workload Offloading

Changing the hardware accelerators on the reconfigurable logic (i.e., using DPR) and offloading computation from software to reconfigurable hardware (i.e., task acceleration) are processes managed by the software code that runs on the embedded processors of the FPGA SoC. This code will be written manually by the application programmers beforehand. The workload management infrastructure proposed in this

[9] https://github.com/des-cei/fpga-monitor/

Thesis, in turn, has the ability to handle autonomously all the hardware acceleration requests coming from a dynamic workload, using a specific scheduling policy to decide when and where to execute each of them on the reconfigurable logic of the FPGA.[10] The scheduling policy can be easily replaced due to the modular design of the infrastructure. For the remainder of this chapter, a First Come, First Served (FCFS) scheduling policy is used for simplicity in the infrastructure description; however, Chapter 4 presents the conflict-aware scheduling policy developed in this Thesis.

The workload management infrastructure is implemented in a two-queue structure that operates in a series of stages, as depicted in Figure 2-8. Those stages are briefly described below:

1. **Arrival stage:** each of the hardware acceleration requests that arrive at the system as part of a dynamic workload is moved to a waiting queue to be later transferred to the reconfigurable logic of the FPGA.

2. **Scheduling stage:** in this stage, a continuous inspection of the waiting queue is carried out. Whenever one of the hardware acceleration requests can be scheduled (i.e., there are reconfigurable slots available, and the scheduling policy dictates that it is the next task to be accelerated), it is transferred to a scheduling queue, including information about where it needs to be offloaded in the reconfigurable logic. It is essential to note that this infrastructure assumes the FPGA fabric is partitioned into reconfigurable slots.

3. **Execution stage:** when a task is moved to the scheduling queue, the workload management infrastructure uses ARTICo^3 primitives to configure the reconfigurable slots dictated by the scheduling policy.[11] The required hardware accelerators are then reconfigured and executed until the task's job size is processed (i.e., the number of iterations that the hardware accelerator must run to process the input data completely). Afterwards, the results are collected and made accessible to the user. Note that the tasks to be accelerated arrive as containers and are deployed following the user/daemon methodology explained in Section 2.3.2.

[10] Note that most hardware acceleration frameworks (e.g., ARTICo^3, AMD Vitis) are originally intended to manage workloads defined at design time.

[11] Even though ARTICo^3 is the hardware accelerator framework selected for this Thesis, this infrastructure is designed in a modular manner to be compatible with other approaches. In fact, it has been successfully tested with other methodologies such as MDC.

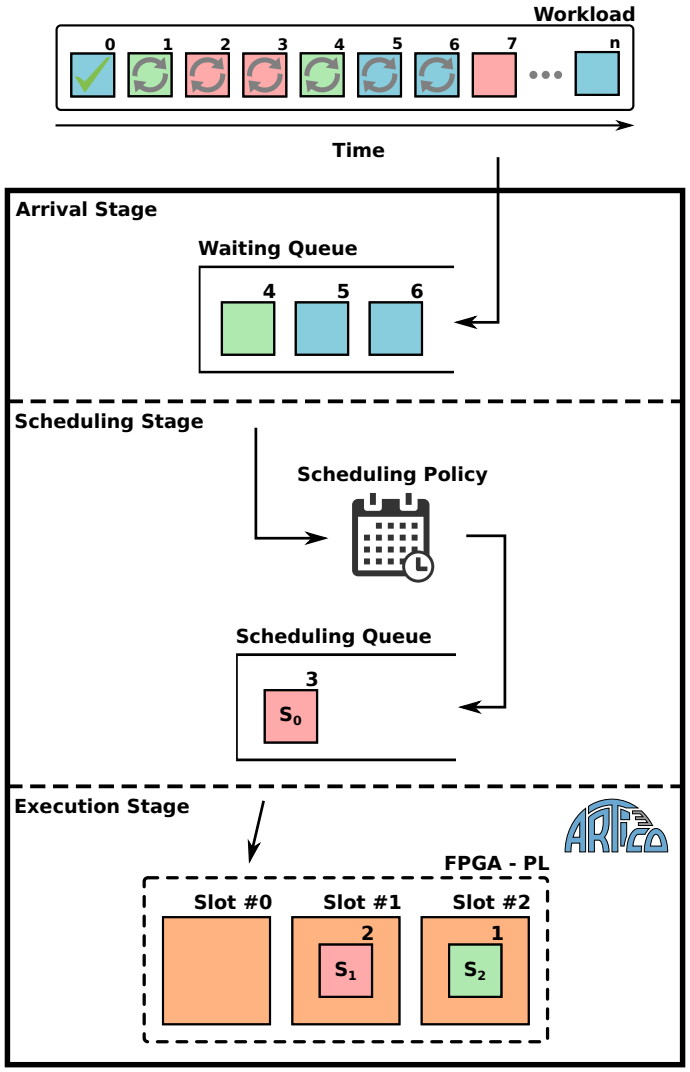


Figure 2-8: Workload offloading process. The symbols in each workload request indicate: green tick symbol, the request has already been executed in hardware; gray loading symbol, the request has arrived and is being managed by the offloading process, and a blank space, the request has not arrived yet. Note that in the scheduling stage, based on the policy, it is decided in which reconfigurable slot the request will be executed, which is indicated with an $S_{\#slot}$.

Workload Registration

This component registers every task that is accelerated during its operation, including relevant related data (e.g., kernel function, job size, number of accelerators, arrival time, start/ready time). Using this information, a map can be generated showing which kernels are executed and where at any given instant throughout the entire workload execution. Figure 2-9 shows a representation of this type of map generated with the workload management infrastructure. Moreover, since the monitoring framework described in Section 2.3.3 has also been integrated into this infrastructure, power consumption and performance traces can be extracted during the execution of the

workload and synchronized with the generated map, which is fundamental to being able to associate the measured traces with the corresponding tasks properly. The yellow lines in Figure 2-9 indicate these trace acquisition events.

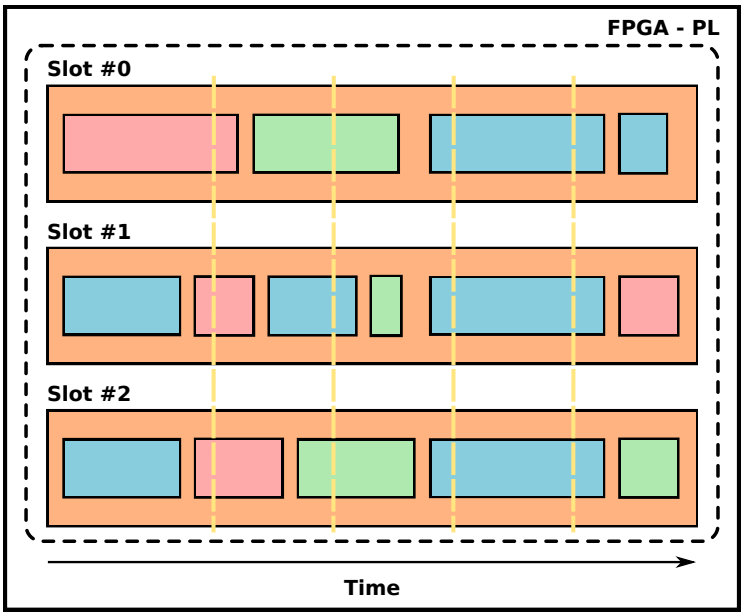


Figure 2-9: Graphic representation of the workload registration map. It represents which kernels have been executed in each slot over time. The yellow lines show the acquisition events.

The integration of the components described so far in this chapter results in a workload management infrastructure capable of accelerating dynamic workloads in the reconfigurable logic of an FPGA, while collecting power consumption and performance traces at run time. Moreover, the task virtualization methodology, together with the porting of ARTICo^3 to cloud FPGAs, enables the implementation of this infrastructure to the diverse set of nodes in the cloud-edge continuum. In the next section, the infrastructure extension to the cloud-edge continuum will be presented.

2.4 Extension to the Cloud-Edge Continuum

While the presented infrastructure enables the implementation of hardware acceleration in any node of the cloud-edge continuum, a meaningful continuum requires the possibility of adding or replacing nodes, with tasks being moved around them with no user intervention. This section extends the infrastructure to enable these capabilities.

2.4.1 Seamless Deployment of Hardware Acceleration Across the Continuum

Workload tasks (i.e., hardware acceleration tasks) are deployed as containers for portability. However, as stated in Section 1.4, the manual run-time management of containers is a tedious, non-scalable, and error-prone process. Therefore, the proposed infrastructure relies on a container orchestrator known as Kubernetes.[12] Kubernetes creates clusters, each one composed of a set of nodes. Every node in the cluster works as a Kubernetes agent, except for the Kubernetes master (which can be either a dedicated node or a node sharing an existing one). The master is responsible for identifying and managing the nodes, as well as deploying containers among them. Meanwhile, each agent serves the master's requests (e.g., container deployments, network management) via a local container runtime (e.g., Docker) running on each node. The continuum nodes are monitored and managed locally by their corresponding agents, reporting the information back to the master.

For example, when a user of a Kubernetes cluster initiates a container deployment, the master selects the most suitable nodes for the deployment and the number of replicas to be deployed. Each of the chosen agents receives a container deployment request from the master, specifying the required replicas. The agents then leverage their respective local container runtimes to deploy and monitor the containers. A series of communication transactions occurs between the master and the agents to ensure the proper execution of containers and resolve any possible failures. Furthermore, container deployments can be rescaled at run time when needed, and event nodes can be added or replaced on demand.

Among the many Kubernetes distributions available, this Thesis relies on K3s, an official lightweight Kubernetes distribution with a reduced set of components.[13] Its minimal footprint makes it ideal for lower-end edge devices that could struggle to run more complex Kubernetes distributions. However, any other Kubernetes distribution would be compatible with the proposed methodology.

Figure 2-10 illustrates a cloud-edge continuum cluster infrastructure that enables transparent deployment of hardware acceleration to a diverse set of nodes. Note that each node presents the corresponding implementation of ARTICo^3, while user applications are deployed as containers.

[12] https://kubernetes.io/
[13] https://k3s.io./

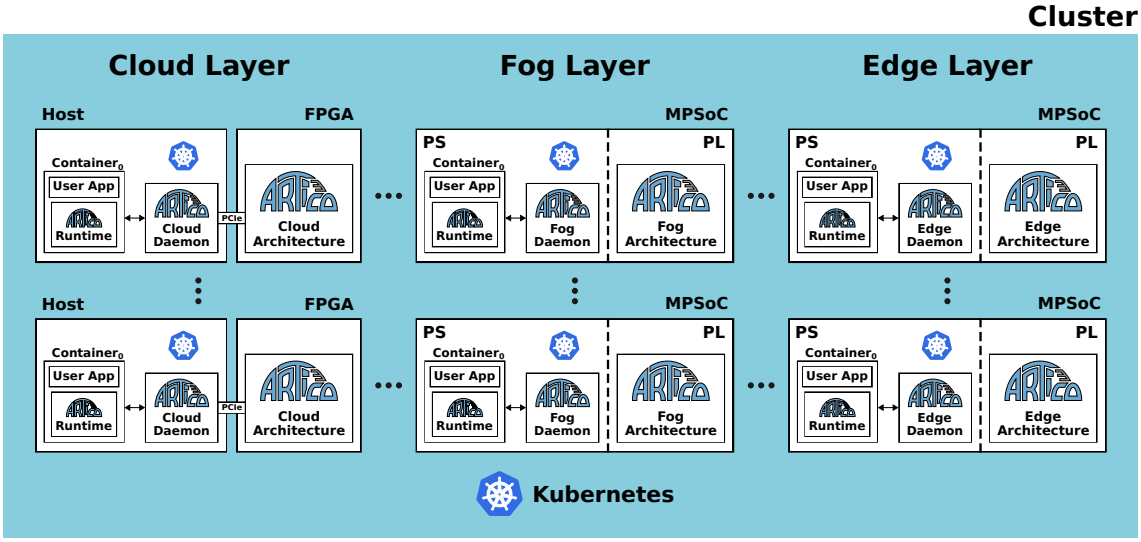


Figure 2-10: Representation of a cloud-edge continuum cluster.

2.4.2 Dynamic Management of Multi-Cluster Topologies

One can observe that the set of nodes in Figure 2-10 has been classified as a cluster, not a continuum. This terminology has been used on purpose. In a real cloud-edge continuum structure, nodes can be placed in geographically remote regions and managed by multiple organizations that cooperate to build a unified continuum of resources. While it might initially seem that creating a unique Kubernetes cluster hosting every node is a good approach, there are several limitations in such environments (e.g., resource isolation, scalability restrictions) [Larsson'20]. The optimal solution is to keep the nodes segregated as separate clusters, each managed by the nodes' owner organizations and/or by local proximity. These clusters can then be connected to form a continuum of resources, comprising multiple clusters.

This Thesis implements this methodology using Liqo [Iorio'23], an open-source tool for the dynamic and seamless management of multiple Kubernetes clusters. Liqo is the selected candidate because it is a mature, well-documented tool that has also been selected for the MYRTUS project, which provides the framework for this Thesis. ArubaKube, the company behind Liqo, is also a member of the consortium.

Building a cloud-edge continuum with Liqo is as simple as running a Liqo process on the master node of each cluster that will form the continuum. Then, clusters can be paired with others, so that each cluster sees its available resources as the sum of its own and those of the others. Deploying a task is then performed as in a typical Kubernetes cluster, considering the new resources native. Behind the scenes, Liqo transparently communicates with the Kubernetes master of the remote cluster to perform the deployment, as if it were an extension of the local cluster. An example of a cloud-edge continuum composed of independent clusters connected via Liqo is illustrated in Figure 2-11.

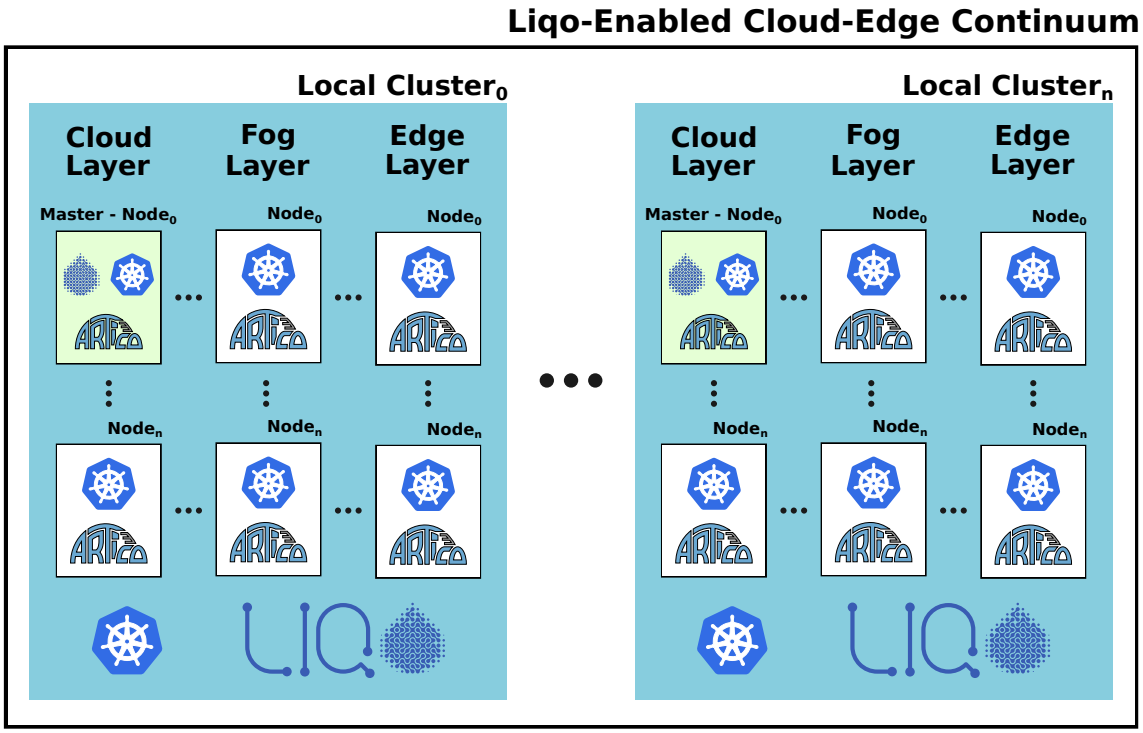


Figure 2-11: Representation of a cloud-edge continuum structure. The master nodes that run the Liqo processes are highlighted in green.

The primary benefits of following this multi-cluster methodology are:

- **Federation:** each cluster maintains its autonomy, and no centralization of synchronization mechanisms between clusters and organizations is required.
- **Isolation:** Liqo does not actually join nodes to a cluster, but virtualizes them as a remote cluster. The security and ownership of the infrastructure are preserved. Organizations do not require full access to each other's nodes.
- **Transparent resource sharing:** workloads are transparently redirected to remote clusters without refactoring applications. Remote nodes are just another available node, expandable in an elastic manner.
- **Low operational overhead:** clusters are modular and manageable, instead of a complex-to-scale flat structure of nodes.

Extending the dynamic workload management infrastructure presented in the previous section with the seamless container management provided by Kubernetes enables hardware acceleration implementation across multiple FPGA nodes. Introducing Liqo on top of that expands its reach to a functional cloud-edge continuum.

Original contribution 2-4 *A virtualization methodology to support the seamless deployment of hardware accelerators across the cloud-edge continuum.*

2.5 Validation

The validation of each component presented in this Thesis is conducted by evaluating the execution of a set of kernels from a well-known HLS benchmark suite. First, this section outlines the benchmark selection criteria for determining the type of kernels to be used for validation. The monitoring framework presented is then thoroughly validated in various configurations. Finally, the proposed workload management infrastructure is evaluated in a real cloud-edge continuum scenario.

2.5.1 Benchmark Evaluation

Benchmarking is an extensively used evaluation approach, enabling a standardized comparison to other alternative solutions. Some well-known examples of benchmark suites are SPEC [SPEC], EEMBC [EEMBC], NPB [Bailey'91], SPLASH [Singh'92], PARSEC [Bienia'08] or MediaBench [Lee'97].

Initially, benchmark suites consisted of a set of applications optimized for a specific platform. However, after the success of parallel computation, the trend shifted towards capturing the requirements shared among different classes of applications by analyzing their communication/computation patterns. The Berkeley Dwarfs classification [Asanovic'06] is one of the most relevant classifications of communication/computation patterns, summarized into the following categories:

- **Dense linear algebra:** involves computations over dense matrices and vectors with regular memory access and high data reuse. Typical in scientific computing, it emphasizes floating-point performance and throughput.
- **Sparse linear algebra:** focuses on operations involving sparse matrices with irregular indirect memory access. Performance is often limited by memory bandwidth and access patterns.

  Spectral methods: employ mathematical transforms such as the Fast Fourier Transform (FFT) to analyze data in the frequency domain. These methods exhibit structured communication and benefit from high arithmetic intensity.
- **N-body methods:** model interactions among many entities, often with pairwise computations that scale quadratically. Some approximation techniques are used to reduce complexity.
- **Structured grids:** use regular grid-based structures for spatial discretization in simulations. Suitable for parallelization due to predictable memory accesses.
- **Unstructured grids:** operate on irregularly connected data points, typical in finite element and finite volume methods. Exhibit irregular data access patterns, posing challenges for optimization and parallelism.

- **MapReduce:** a data-parallel programming model that separates computation into independent map and aggregation-based reduce phases. Widely used in distributed systems for large-scale data processing.
- **Combinational logic:** consists of stateless operations over bits, such as logical functions, hashing, and encryption. Highly parallelizable, it emphasizes logic depth over data movement.
- **Graph traversal:** traverses nodes and edges of graphs. Characterized by irregular control flow and memory access, often poses load-balancing challenges.
- **Dynamic programming:** solves complex problems by recursively breaking them into overlapping subproblems. Requires careful handling of data dependencies and efficient memoization strategies.
- **Backtrack and branch-and-bound:** search-based techniques for constraint satisfaction and combinatorial optimization. Employ pruning strategies to reduce the search space and computational effort.
- **Graphical models:** represent probabilistic relationships through graph-based structures such as Bayesian or Markov networks. Require computationally intensive inference and learning procedures.
- **Finite State Machine:** model systems with a finite number of discrete states and transitions based on inputs. Used in control logic and protocol processing with predictable behavior.

Modern benchmarking suites often target the complete set of Berkeley Dwarfs, enabling thorough analysis of emerging computing architectures. Some efforts have introduced unified cross-platform benchmarks using OpenCL to support CPUs, GPUs, and FPGAs [Krommydas'16]. However, OpenCL lacks performance portability, making it less suitable for evaluating accelerator-centric systems like the one targeted in this Thesis.[14] Hence, device-specific implementations of these Dwarfs remain relevant.

For the validation campaigns of this Thesis, only C-based HLS-oriented benchmark suites have been considered, which typically serve two primary purposes: evaluating HLS tools and supporting architectural exploration in accelerator-centric designs. Keeping the focus on the latter, which is the relevant usage for this Thesis, three suites have been considered: CHStone [Hara'08], MachSuite [Reagen'14], and Rosetta [Zhou'18]. A comparison is provided in Table 2-5.

CHStone was one of the first HLS-oriented benchmark suites, featuring 12 applications ranging from double-precision arithmetic to cryptographic primitives like AES and SHA. Rosetta, a more recent suite, includes six complex applications (e.g., 3D rendering, face detection) designed for exploring both compute- and memory-bound

[14] OpenCL guarantees functional portability across compatible devices, but performance still requires manual tuning due to hardware heterogeneity.

Table 2-5: HLS-oriented benchmark suites.

	CHStone [Hara'08]	**MachSuite [Reagen'14]**	**Rosetta [Zhou'18]**
# Benchmarks	12	19	6
Complexity	Low/Medium	Medium	High
Dwarf-based	No	Yes	No

scenarios. MachSuite offers 12 kernels, with some variants, for a total of 19 Dwarf-oriented benchmarks, positioned in complexity between CHStone and Rosetta.

MachSuite is the most suitable for this Thesis, offering a balanced trade-off between computational demands and resource usage. Table 2-6 details the benchmarks included in MachSuite and the Berkeley Dwarfs they target.

Table 2-6: MachSuite benchmarks [Reagen'14].

Name	**Description**	**Berkeley Dwarf**
AES/AES	AES encryption	Combinational logic
BACKPROP/BACKPROP	Neural Network training	Unstructured grids
BFS/BULK	Breadth-first search (data-oriented)	Graph traversal
BFS/QUEUE	Breadth-first search (expanding-horizon)	Graph traversal
FFT/STRIDED	Fast Fourier Transform (recursive)	Spectral methods
FFT/TRANSPOSE	Fast Fourier Transform (two-level)	Spectral methods
GEMM/NCUBED	Matrix multiplication (naive)	Dense linear algebra
GEMM/BLOCKED	Matrix multiplication (block-based)	Dense linear algebra
KPM/KPM	String matching	Finite state machines
MD/KNN	Molecular dynamic (k-nearest neighbors)	N-body methods
MD/GRID	Molecular dynamic (spatial decomposition)	N-body methods
NW/NW	DNA alignment	Dynamic programming
SORT/MERGE	Sorting (mergesort)	MapReduce
SORT/RADIX	Sorting (block comparison)	MapReduce
SPMV/CRS	Sparse matrix/vector multiplication	Sparse linear algebra
SPMV/ELLPACK	Sparse matrix/vector multiplication	Sparse linear algebra
STENCIL/STENCIL2D	Stencil computation (2D)	Structured grids
STENCIL/STENCIL3D	Stencil computation (3D)	Structured grids
VITERBI/VITERBI	Hidden Markov model estimation	Graphical models

For the evaluations performed during this manuscript, the MachSuite benchmarks have been implemented in ARTICo^3 for a diverse set of FPGA devices during this Thesis (i.e., Pynq-Z1,[15] Zynq US+ ZCU102,[16] Alveo U250,[17] Kria KV260 SoM,[18] Genesys 2[19]). Note that eight algorithms were excluded from the experiments either because

[15]XC7Z020-1CLG400C device.
[16]XCZU9EGFFVB1156-2-I device.
[17]A-U250-P64G-PQ-G device.
[18]SK-KV260-G device.
[19]XC7K325T-2FFG900C device.

they exceeded the logic resources available on an ARTICo^3 slot, or because their execution is so long that the traces that would be generated with the monitoring infrastructure do not fit in the reconfigurable fabric when implemented in the smaller FPGAs.[20] The MachSuite benchmarks excluded from the experiments are colored gray in Table 2-6. Note that the CRS kernel has had to be excluded from the AMD Alveo board due to platform-specific implementation incompatibilities; hence, this kernel is not represented in the evaluations related to that specific board. Nonetheless, 9 out of the 12 Berkeley Dwarfs are still represented, ensuring sufficient kernel diversity.

Please note that the validation strategy of this Thesis focuses on functional evaluation rather than performance optimization. Therefore, no HLS-specific directives or code refactoring were applied to the original benchmarks. Only minimal changes were made to adapt the kernels to the ARTICo^3 interface, including the use of 32-bit data types, port flattening (replacing structs with individual elements), and port packing (grouping small ports to optimize DMA transfers).

In some exceptional cases, an additional set of applications may be added to the validation process due to specific characteristics in the corresponding validation campaign (e.g., monitoring framework validation). In any case, it will be indicated when applicable.

2.5.2 Monitoring Framework

The validation of the proposed monitoring framework focuses on its flexibility and compatibility, which are the main drivers behind its design. To do so, this section evaluates a set of monitoring compositions, combining the components shown in Table 2-4, ensuring each is represented in at least one of the instances.

Table 2-7 lists the configurations under evaluation. A reduced set of benchmarking applications has been executed and monitored with the proposed configurations to perform the evaluation (see Table 2-8). Some of them come from the MachSuite benchmarks, others are tailored to the specific hardware acceleration framework in use, such as the Block Matrix Multiplication (BMM) example[21] in ARTICo^3 and a data-flow Convolutional Neural Network (CNN) for MDC. Please note that the purpose of this section is to evaluate the monitoring framework, not the algorithms. A graphical representation of the power and performance traces obtained while running the validation tasks will be shown for each configuration. Each task runs different iterations to ensure the representation can be appropriately visualized. Additionally, information about induced overhead (measured as the average of 1,000 task executions) and resource utilization will be shared and discussed in every case.

[20] Lower-end boards used for edge computing need the implementation of reconfigurable slots with limited resources, constraining the size of the kernels to be implemented in them.

[21] https://des-cei.github.io/tools/artico3/tutorials/matmul/

Table 2-7: Monitoring configurations to be evaluated.

#	Platform	OS	Power	Performance	Control	Acceleration
1	Pynq-Z1	Custom Linux	External board	Internal HW signals	PS	ARTICo^3
2	Kria KV260 SoM	Yocto-Linux	On-chip ADCs	Internal HW signals	PS	MDC
3	Alveo U250	Custom Linux	Platform-specific	Internal HW signals	PCIe	ARTICo^3
4	Zynq US+ ZCU102	Petalinux	External board	AXI bus	PS	AMD Vitis
5	Genesys 2	Bare-metal	On-chip ADCs	Internal HW signals	PL	ARTICo^3

Table 2-8: Hardware acceleration tasks used for validation.

Name	Description
BMM	A blocked version of matrix multiplication [Rodríguez'18]
FFT	Recursive formulation of the Fast Fourier Transform [Reagen'14]
CNN	A convolutional neural network [Manca'25]
AES	A common block cipher [Reagen'14]

Configuration #1

The Pynq-Z1[22] platform features a Zynq 7000 low-end reconfigurable device suitable for edge computing environments. Hardware acceleration is implemented with ARTICo^3, and the system is controlled from the PS under a custom Linux. The performance traces monitored are internal start/ready signals from the accelerators. Since no on-chip ADCs are available, power consumption is measured with the external measurement board directly on the board's power supply. Figure 2-12 and Figure 2-13 show the power consumption (top) and performance traces (bottom) of the BMM and FFT, respectively, each with four replicas (i.e., four dedicated accelerators).

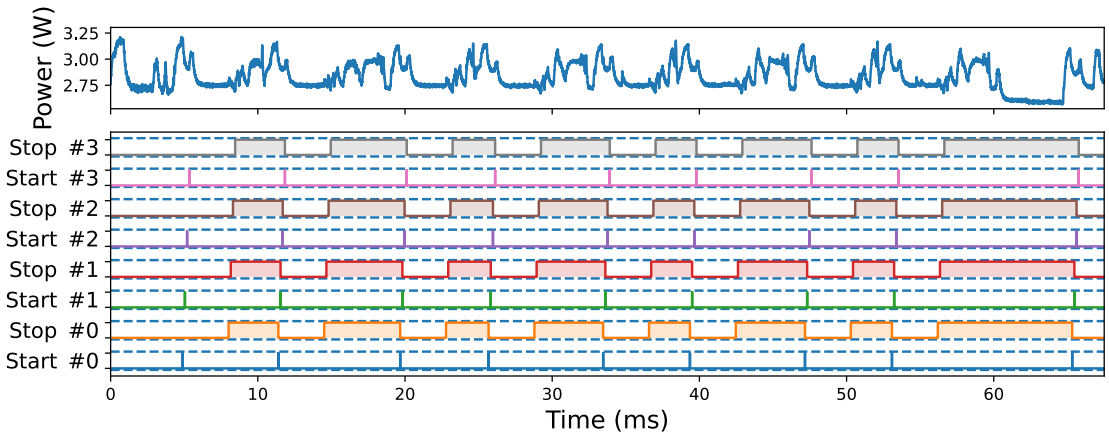


Figure 2-12: Power and performance monitoring running the BMM with configuration #1.

The high sampling frequency of the external measurement board (up to 1 Msample/s) renders detailed power consumption waveforms. This high resolution

[22] XC7Z020-1CLG400C device.

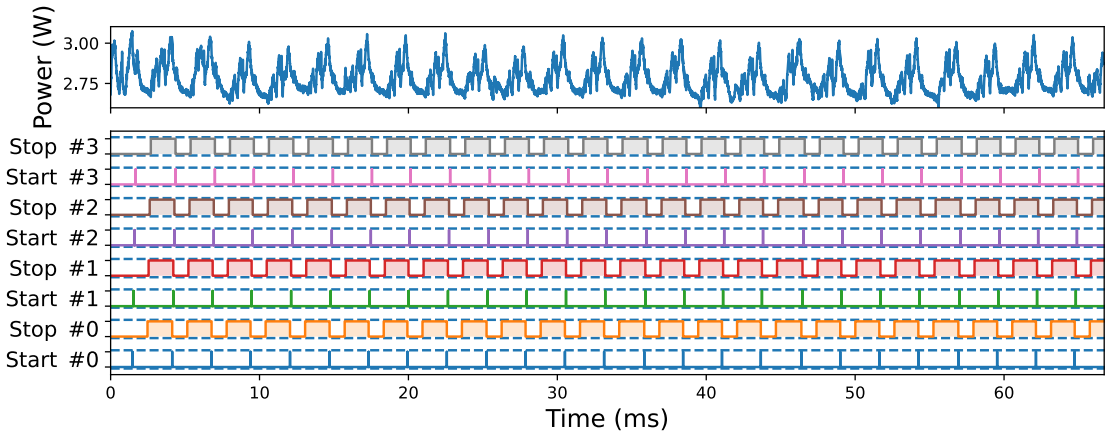


Figure 2-13: Power and performance monitoring running the FFT with configuration #1.

enables a deeper analysis of the system's power consumption, which is crucial in power-aware applications.

Figure 2-14 shows a monitoring snapshot of the BMM execution, with each stage of the process highlighted. In this example, the multiplied matrices are 512×512 elements, and each accelerator processes a 64×64 block. With four accelerators available, two rounds (eight partial multiplications) are needed to produce eight 64×64 submatrices, which are then accumulated element-wise to generate a single 64×64 block of the final solution.

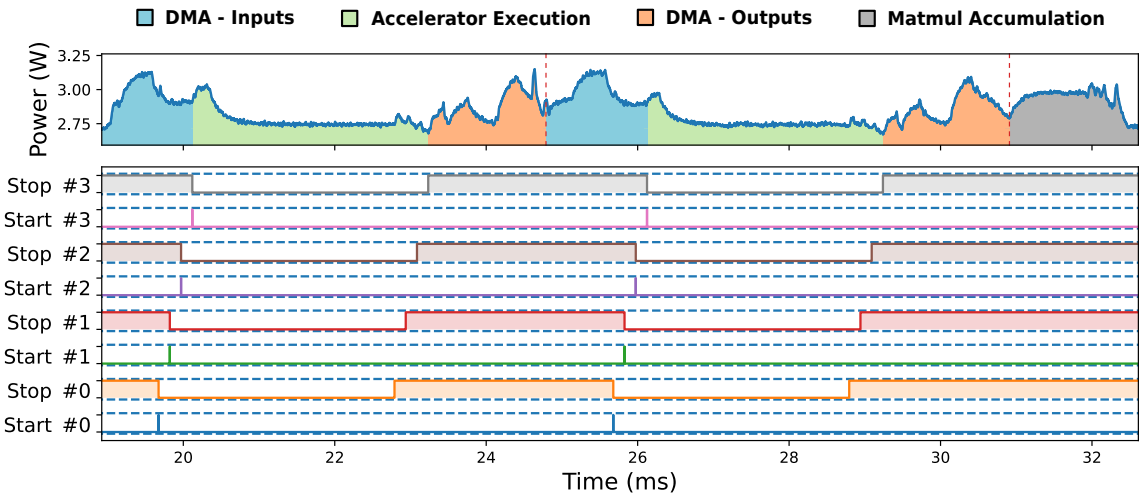


Figure 2-14: Detailed acceleration stages of a BMM execution captured with the monitoring framework on configuration #1.

In the first round, the input matrices are transferred to the accelerators via DMA (blue region), resulting in high power consumption since the CPU handles the transfer. Then, hardware acceleration begins, marked by the rising edges of the start signals (green region). During this stage, power consumption drops as the CPU remains idle.

Once all accelerators finish, indicated by the rising edges of the stop signals, the CPU retrieves the results via DMA (orange region), again increasing power consumption.

This completes one round of four accelerator executions. Since two rounds are required to obtain the eight 64×64 submatrices for accumulation, the same sequence is repeated. After the second round, the CPU performs the elementwise accumulation (gray region), causing another peak in power consumption and producing a single 64× 64 block of the final result. To compute the complete 512×512 output matrix, this process must be repeated a total of 16 times.

Such a high sampling frequency, which enables detailed power analysis, comes at the expense of resources (mainly BRAMs), as observed in Table 2-9. Fine-tuning the number of samples to collect will allow the user to explore trade-offs between area usage and monitoring window size. Finally, Table 2-10 shows the overhead induced on the system. Since trace capture is performed entirely in hardware, it has no impact on the system's performance. However, captured data must be transferred from BRAMs to the processor, which may be affected differently based on the application. For instance, since the FFT is much faster than BMM, more traces are generated per time unit, inducing higher relative overheads.

Table 2-9: Config. #1 - Resource utilization.

Power Traces	65,536 samples
Performance Traces	8 probes 16,384 samples[1]
LUTs	1,560
FFs	734
BRAMs	44

[1] A 32-bit timestamp stored with every sample

Table 2-10: Config. #1 - Monitor overhead.

Application	**Overheads**[1]	
	Capture	**Transfer**
BMM	0%	0.72%
FFT	0%	11.11%

[1] Monitoring time expressed as a percentage of total application execution time

Configuration #2

A more performing Zynq UltraScale+ device, embedded in a Kria KV260 platform,[23] is employed in this configuration. Here, the application domain changes, enabling the acceleration of dataflow tasks with MDC. The on-board ADCs of this platform enable power consumption measurement right out of the box. Performance is again measured from internal start/ready signals. A Yocto-based Linux on the PS controls the system.

Figure 2-15 and Figure 2-16 show the monitored traces when running a one-replica CNN and AES tasks, respectively. The faster kernels and the lower sampling rate of on-chip ADCs render less descriptive power consumption waveforms than before. But still, significant power variations can be detected, in this case, between hardware acceleration (i.e., when start/ready signals toggle) and idle.

[23] SK-KV260-G device.

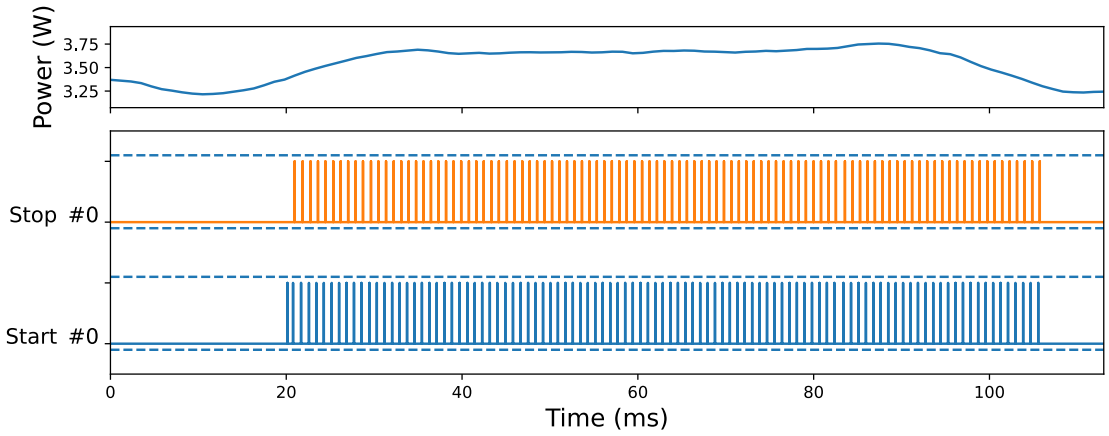


Figure 2-15: Power and performance monitoring running the CNN with configuration #2.

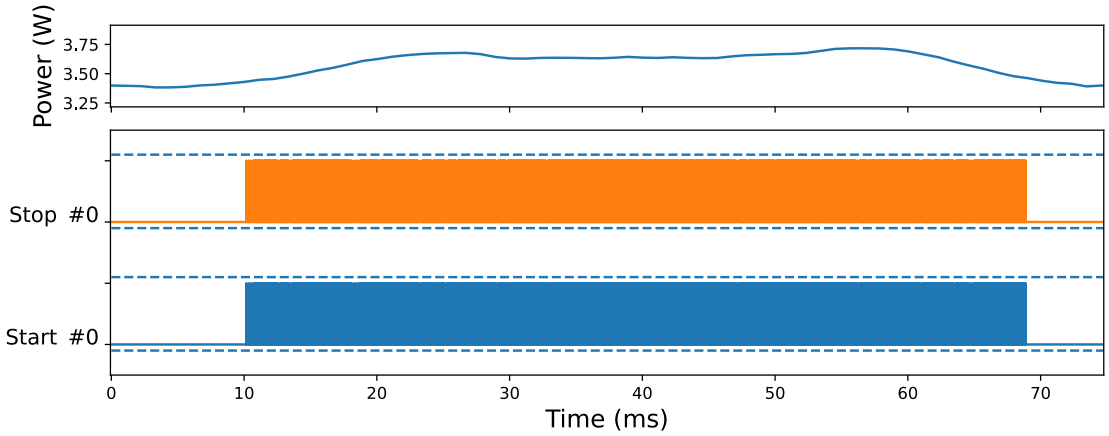


Figure 2-16: Power and performance monitoring running the AES with configuration #2.

Resource-wise, this configuration is particularly lightweight (see Table 2-11). Since power traces are measured from SW, BRAMs stores just performance traces. Regarding overhead, the fact of reading on-chip ADCs from SW makes the capture overhead non-negligible, as shown in Table 2-12. However, the results have shown a static overhead independent of the number of traces to be read; therefore, this approach will benefit significantly from longer capture windows. Indeed, the CNN application, which is slightly longer, faces a smaller capture overhead. Transfer overhead is negligible since only performance traces are transferred to the processor, and these traces, usually start/ready signals, tend to be relatively few.

Table 2-11: Config. #2 - Resource utilization.

Power Traces	Software
Performance Traces	2 probes 16,384 samples[1]
LUTs	1,046
FFs	397
BRAMs	15.5

[1] A 32-bit timestamp stored with every sample

Table 2-12: Config. #2 - Monitor overhead.

Application	**Overheads**[1]	
	Capture	**Transfer**
CNN	10.17%	0.25%
AES	13.97%	0.30%

[1] Monitoring time expressed as a percentage of total application execution time

Configuration #3

This configuration features an Alveo U250 platform,[24] a high-end fog/cloud computing device. It consists of an FPGA attached to a host PC via a PCIe connection, proving that the monitoring framework is compatible with any other PCIe-based platform. A custom Linux from the host PC controls the overall system. Like the first example, ARTICo^3 has been used for acceleration implementation, in this case, accelerating 16-replica BMM and FFT (see Figure 2-17 and Figure 2-18), exploiting the abundance of resources. Notice that the performance plotting, measured from internal signals, has been limited to two replicas for clarity. Another key difference here is the measurement of power consumption. Since this platform does not feature on-chip ADCs, a platform-specific solution, known as Card Management Solution (CMS) subsystem [Xilinx'23a], has been employed, demonstrating the adaptability of the proposed monitor to other commercial measurement infrastructures. It is curious how the evolution of the power consumption does not correlate with the performance signals. The reason is the high static and dynamic power consumption of the platform (up to 20W), which makes power variations due to hardware execution less relevant. This is a good example highlighting the importance of acquiring synchronized power/performance traces to enable an efficient continuum.

Table 2-13 shows that resource utilization is similar to previous cases. However, this platform is ideal for longer measurement windows due to its massive resources available. The capture overhead shown in Table 2-14 is roughly 1%, coming from the periodic access to the CMS. In turn, the data transfer overhead is diluted on large application executions.

[24] A-U250-P64G-PQ-G device.

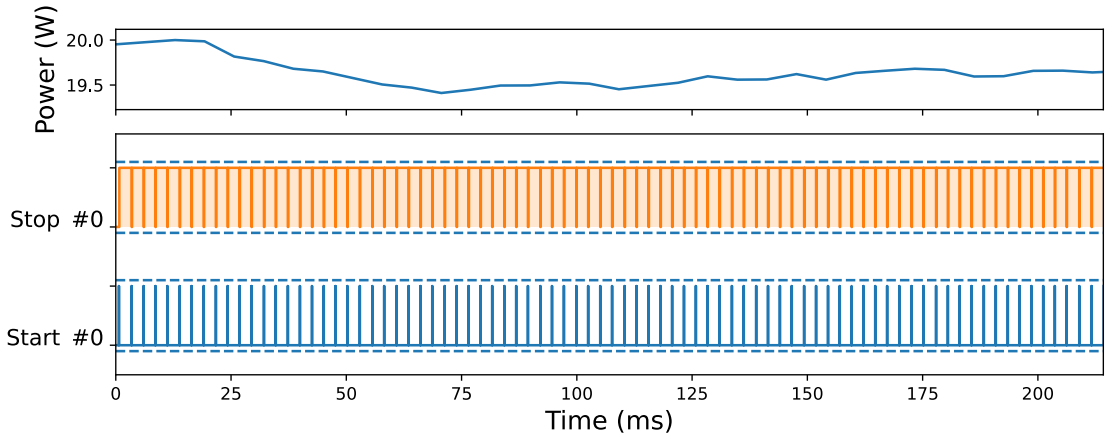


Figure 2-17: Power and performance monitoring running the BMM with configuration #3.

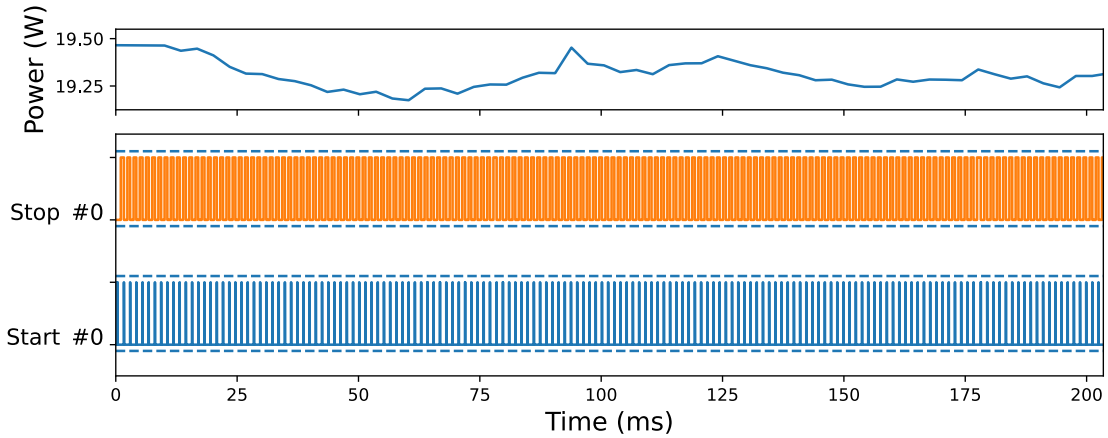


Figure 2-18: Power and performance monitoring running the FFT with configuration #3.

Table 2-13: Config. #3 - Resource utilization.

Power Traces	Software
Performance Traces	32 probes 16,384 samples[1]
LUTs	1,044
FFs	468
BRAMs	29

[1] A 32-bit timestamp stored with every sample

Table 2-14: Config. #3 - Monitor overhead.

Application	**Overheads**[1]	
	Capture	**Transfer**
BMM	1.2%	0%
FFT	0.91%	0%

[1] Monitoring time expressed as a percentage of total application execution time

Configuration #4

A Zynq UltraScale+ is used in this configuration via a ZCU102 development board.[25] This configuration aims to prove the compatibility of the monitor with the acceleration flow of AMD Vitis, the major commercial hardware acceleration framework. Hence, the platform is managed from the PS under an OS built with Petalinux (i.e., AMD's embedded Linux framework). The accelerators created with AMD Vitis are controlled via AXI buses using AMD's acceleration runtime (i.e., XRT).

Traces captured from a two-replica FFT are shown in Figure 2-19, proving the compatibility of the monitoring framework with this acceleration approach. The user can specify any particular combination of signals to be monitored from an AXI bus. Then, the monitor can be configured to identify specific communication bus events among those signals (such as start/stop commands), generating 1-bit representations of them to be used as performance traces. For this configuration, the external measurement board has been leveraged to gather power consumption measurements from dedicated shunt resistors on the PS and PL of the platform. Please note the spikes in PS power consumption between consecutive stop and start events, which correspond to the data transfers from/to accelerators that occur during those events. Similarly, a significant drop in power consumption can be observed after the last stop signal (i.e., end of hardware acceleration). PL power consumption, in contrast, faces minimal variability since the accelerators are already configured on the entire measurement window.

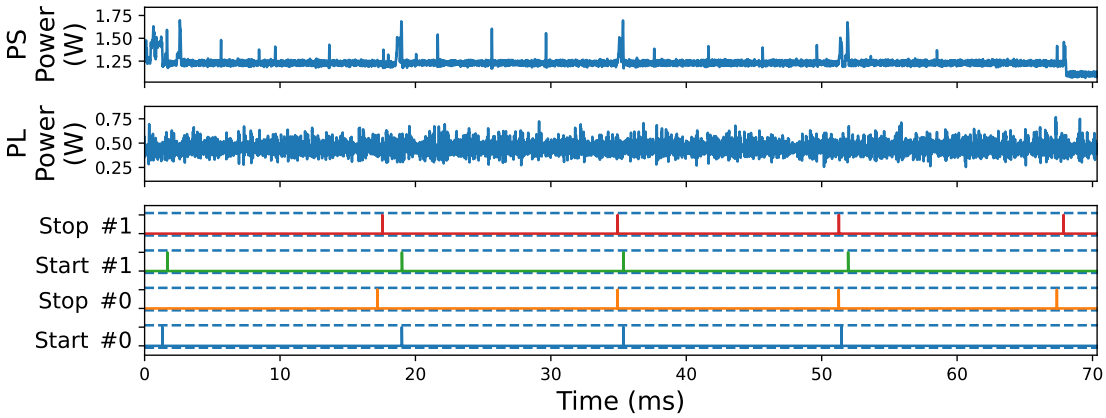


Figure 2-19: Power and performance monitoring running the FFT with configuration #4.

The analysis of resource utilization (Table 2-15) and induced overhead (Table 2-16) is similar to that of configuration #1, since the trace collection methods are equivalent.

[25] XCZU9EGFFVB1156-2-I device.

Table 2-15: Config. #4 - Resource utilization.

Power Traces	131,072 samples
Performance Traces	32 probes[1] 4,096 samples[2]
LUTs	2,179
FFs	925
BRAMs	52

[1] Extracted from the AXI bus signals
[2] A 32-bit timestamp stored with every sample

Table 2-16: Config. #4 - Monitor overhead.

Application	**Overheads**[1]	
	Capture	**Transfer**
FFT	0%	9.00%

[1] Monitoring time expressed as a percentage of total application execution time

Configuration #5

A Kintex-7 family FPGA is featured in this configuration as a Genesys 2 platform.[26] The main variation is the use of a RISC-V soft-core processor (CVA6 [Zaruba'19], from OpenHW Group) configured on the PL to control the platform in bare-metal, showing the monitor is compatible with soft-core processors from diverse architectures, as well as bare-metal implementations.

Figure 2-20 shows the execution of a four-replica BMM implemented with ARTICo^3. The performance traces are obtained from internal HW signals. The power consumption traces, conversely, are gathered from on-chip ADCs. However, contrary to configuration #2, the monitor manages the process from HW since no I²C manager is available on the platform due to the lack of a PS.

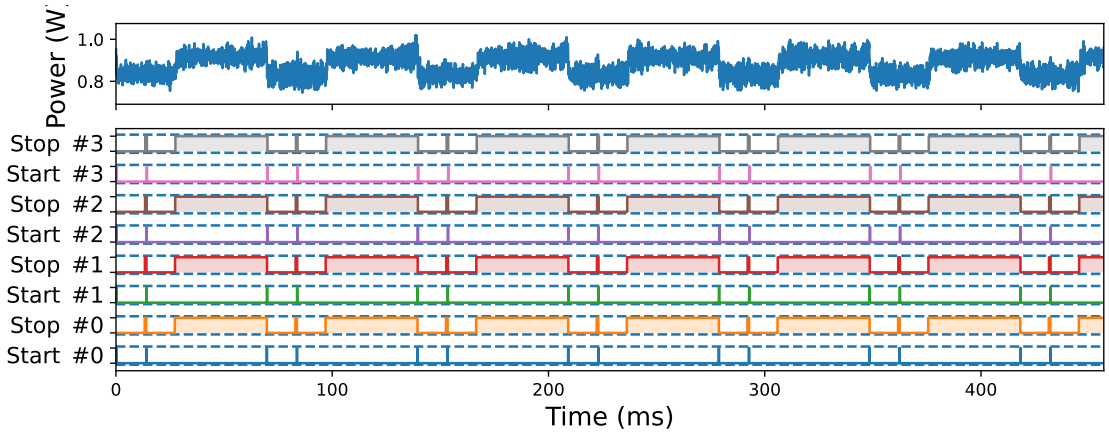


Figure 2-20: Power and performance monitoring running the BMM with configuration #5.

The resources are utilized similarly to configuration #1 and configuration #4 as shown in Table 2-17. Regarding the overhead (see Table 2-18), since power is captured with I²C, the traces generated are fewer than with the SPI approach, inducing negligible

[26] XC7K325T-2FFG900C device.

transfer overheads. Moreover, there is no run-time capture overhead, as opposed to configuration #2, since the I²C manager used to measure power consumption is implemented in hardware in this case.

Table 2-17: Config. #5 - Resource utilization.

Power Traces	32,768 samples
Performance Traces	8 probes 16,384 samples[1]
LUTs	1,593
FFs	720
BRAMs	36

[1] A 32-bit timestamp stored with every sample

Table 2-18: Config. #5 - Monitor overhead.

Application	**Overheads**[1]	
	Capture	**Transfer**
BMM	0%	0.08%

[1] Monitoring time expressed as a percentage of total application execution time

Accuracy and Synchronization of the Traces

The Configuration #1 from Table 2-7 has been used to evaluate the accuracy of the obtained power measurement and trace synchronization. With this aim, a BMM accelerator with a single replica is executed, and power consumption (board power supply) and performance traces (start/ready signals) are gathered with the monitoring infrastructure. Simultaneously, the same power consumption (from an exposed shunt resistor) and performance traces (a crafted signal exposed to the oscilloscope, indicating when the kernel is running) have been captured with an external oscilloscope.

Figure 2-21 displays the traces obtained using the monitoring infrastructure. Conversely, Figure 2-22 shows the data gathered with the oscilloscope. Note that no processing has been applied to the measurements of the oscilloscope apart from converting from shunt resistor voltage to power. The figures show that traces obtained with the proposed monitoring infrastructure present minimal differences from those obtained with the oscilloscope, demonstrating high precision in power consumption measurement and trace synchronization.

Results Discussion

The results demonstrate that the proposed monitoring framework is highly adaptable, supporting diverse platforms and enabling detailed performance and power analysis. It allows users to balance trade-offs between power detail, resource usage, and system performance. Different platforms can be paired with the desired OS, allowing tailored configurations to monitor hardware acceleration across various domains managed by the processor of choice. The external measurement boards can provide fine-grain power monitoring at the cost of resource usage, while on-chip ADCs offer more

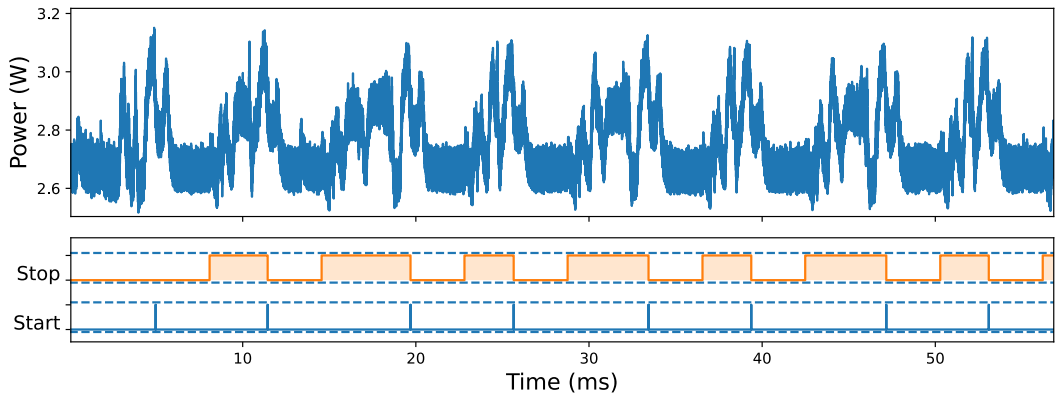


Figure 2-21: Accuracy and trace synchronization - Monitored traces.

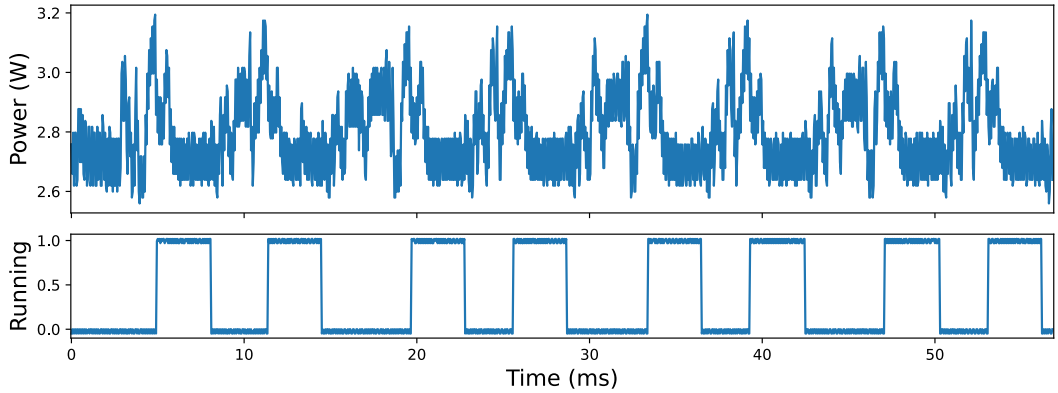


Figure 2-22: Accuracy and trace synchronization - Oscilloscope measurements.

straightforward, less resource-demanding power measurements. The minimal capture overhead of the framework ensures low system impact, with transfer overheads varying based on application speed and capture window length. Additionally, further customization can be applied based on monitoring needs or limitations. For example, users can select more probes and trace samples for debugging or opt for a lightweight, lower-resolution approach for run-time processing. The monitoring framework offers an efficient and flexible solution to most hardware acceleration monitoring needs.

Conclusions

A modular and composable framework for monitoring performance and power consumption in reconfigurable FPGA-based systems has been introduced. Its architecture leverages platform-agnostic interfaces to seamlessly integrate target-specific components (e.g., acceleration frameworks, ADCs, CPU cores, and FPGA boards), enabling flexible plug-and-play monitoring and tracing capabilities across diverse user-defined scenarios.

The experimental results demonstrate that the proposed framework can be tailored to various application requirements. It supports lightweight, low-overhead monitoring for real-time systems and high-resolution data acquisition through dedicated external hardware for precise execution profiling. One of the possible uses for this profiling capability, which will indeed be described in Chapter 3, is to facilitate data-driven modeling, prediction, and estimation of key run-time metrics, enhancing the adaptability and performance analysis of each node in the continuum.

Note that this monitoring framework will play a significant role in the following chapters as part of the modeling and scheduling methodologies of this Thesis.

2.5.3 Cloud-Edge Continuum Infrastructure

This section presents an evaluation of the proposed cloud-edge continuum infrastructure. First, the experimental environment is outlined, followed by an in-depth analysis of the resulting performance data.

Experimental Setup

An implementation has been built to evaluate the proposed cloud-edge infrastructure. Although the system reflects the architecture shown in Figure 2-10, it has been reduced to a single representative node per layer (i.e., cloud, fog, edge) for the sake of demonstration, as depicted in Figure 2-23. Each node being a distinct device with progressively greater reconfigurable resources, power consumption, and cost to reflect the typical layers in the continuum. Despite this simplification, the integration of Kubernetes and the inclusion of virtualization mechanisms (i.e., user/daemon execution model, containerization) make it straightforward to scale the infrastructure when needed.

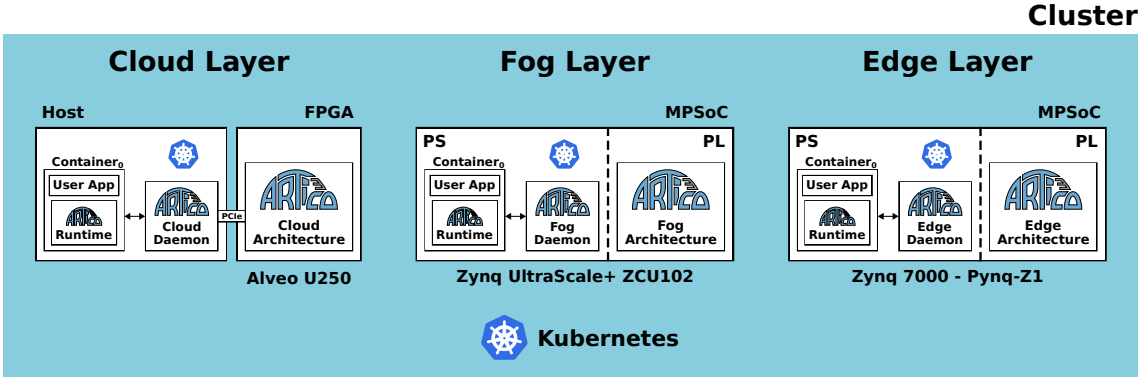


Figure 2-23: Real cloud-edge continuum cluster used for the experimental validation.

The detailed hardware configuration of each FPGA-based system involved in the evaluation is listed in Table 2-19. Please note that the cloud environment is powered by an Intel Xeon Gold 6338 CPU.

Table 2-19: Cloud-edge continuum experimental setup.

Layer	Board (FPGA device)	Frequency		# Reconf. Slots (LUTs per Slot)
		CPU	FPGA	
Cloud	Alveo U250 (A-U250-P64G-PQ-G)	2 GHz	100 MHz	16 (44.1K)
Fog	Zynq UltraScale+ ZCU102 (XCZU9EG-2FFVB1156)	1.2 GHz	100 MHz	8 (21.6K)
Edge	Pynq-Z1 (XC7Z020-1CLG400C)	650 MHz	100 MHz	4 (6.4K)

The allocation of reconfigurable slots differs in size (e.g., LUTs) between platforms due to the varying availability of resources. As detailed in Table 2-19, the number of slots on each platform has been configured as consecutive powers of two. This approach simplifies comparisons despite significant differences in available resources between devices.

To ensure a realistic performance assessment, the MachSuite benchmarks described in Section 2.5.1 have been used as the hardware acceleration task in this validation campaign. In addition, the matrix multiplication example from the ARTICo^3 documentation has been included to broaden the scope of the experiments.

Although the MachSuite benchmarks have been kept unoptimized since this test focuses on validation rather than acceleration, the matrix multiplication example was optimized on the cloud platform to showcase the benefits of vast resource utilization. In this case, loop unrolling was applied to better exploit parallelism and leverage available hardware; higher degrees of unrolling enable better utilization of each reconfigurable slot.

Experimental Results

The benchmark applications have been accelerated on each system, with the number of accelerators corresponding to the number of reconfigurable slots available per platform, as shown in Table 2-19. In this implementation, every component of the proposed workload management infrastructure has been used. The workload containerization has been performed using Docker, enabling platform-independent deployment. The deployment process across continuum nodes is based on Kubernetes, while ARTICo^3 handles accelerator management at the hardware level. Finally, the performance metrics have been captured with the monitoring framework. Please note that since there is only one cluster under test, Liqo is not used. A multi-cluster continuum scenario managed with Liqo will be explored in Chapter 5, as that chapter integrates all components of this Thesis and offers a more meaningful analysis of the benefits and trade-offs of using multiple clusters.

The execution time to accelerate each application in each continuum node is illustrated in Figure 2-24, normalized to the fastest execution time (i.e., the cloud). Since this figure itself validates the proper operation of the proposed cloud-edge continuum infrastructure, the rest of the validation section focuses on analyzing the available trade-offs observed when moving tasks across the continuum.

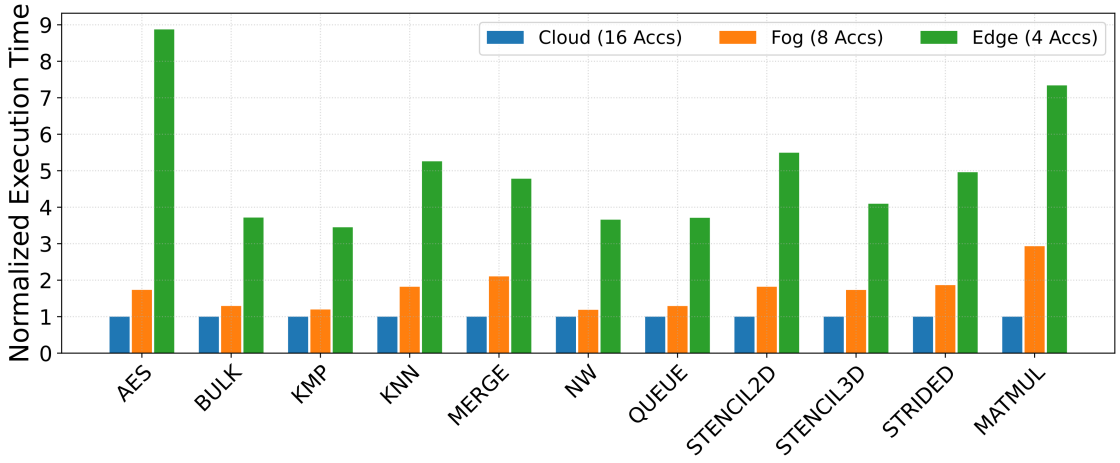


Figure 2-24: Task execution time per platform, normalized to the cloud.

The normalized execution time progression across platforms is consistent for most MachSuite kernels due to their equivalent implementation. Variations in performance are primarily due to differences in the number of available parallel accelerators: 16 on the cloud, 8 on the fog, and 4 on the edge. However, there are also differences due to host processor capabilities and data transfer overheads. Performance can vary significantly depending on whether a kernel is compute-bound, with parallelism having a greater impact on acceleration(e.g., KNN, STRIDED), or memory-bound, with parallelism having a lesser impact on acceleration (e.g., NW, QUEUE).

The matrix multiplication application, having undergone performance optimization, has demonstrated a notable speedup on the cloud platform. This highlights the advantages of performance optimization in combination with a larger number of parallel accelerators, a typical approach used in cloud devices.

Table 2-20 provides a summary of available resources across platforms. It also includes the theoretical maximum number of reconfigurable slots based on a standardized slot size derived from the constraints on the edge node.[27] This uniform slot size is chosen to simplify the portability of the hardware between devices. As will be briefly explored in Chapter 5, it could be considered the use of larger slots with deeper kernel parallelization for larger platforms to better leverage the capabilities of each device type.

Based on resource availability, the cloud and fog platforms could support up to 32 times and 5 times more accelerators than the edge, respectively. This implies that

[27]Each slot area is limited by the size of a clock region, which on edge devices is inherently smaller, so the maximum reconfigurable slot is constrained by these physical clock-region dimensions.

Table 2-20: Resource available per platform.

Platform	LUTs	FFs	DSPs	BRAMs	Maximum # of Reconfigurable Slots
Cloud	1,707,232	3,456,000	12,288	2,688	≈ 256 (32×)
Fog	274,080	548,160	2,520	912	≈ 42 (5.25×)
Edge	53,200	106,400	220	140	≈ 8 (1×)

higher performance through increased parallelism (i.e., more kernel replicas) can be achieved as workloads move closer to the cloud. Additionally, this abundance of resources opens opportunities for shared multi-tenant usage, improving infrastructure utilization and distributing the associated costs.

Nonetheless, migrating workloads to the cloud introduces trade-offs in terms of energy and latency. Power consumption measurements, detailed in Table 2-21, show that cloud systems draw significantly more power than other alternatives (note that although cloud and fog power consumptions are similar during these tests, power consumption is expected to scale with resource utilization in the cloud). This holds when focusing solely on power consumption, as edge devices generally offer better energy efficiency. However, this assessment does not take into account the energy or cost associated with transferring data from edge to cloud. As a result, deciding where to offload or accelerate each application involves carefully weighing the performance benefits against the power budget constraints. Please note that power consumption in fog and edge devices has been measured at the power supply. For the cloud platform, the monitoring framework leverages the onboard sensors to measure the power drawn.

Table 2-21: Power consumption and reconfiguration time per platform.

Platform	Power Consumption	Reconfiguration Time per MB of bitstream
Cloud	23 W (6.57×)	2,883 ms (1×)
Fog	23 W (6.57×)	3.567 ms (1.24×)
Edge	3.5 W (1×)	19.242 ms (6.67×)

Another critical aspect is the reconfiguration time, which is also provided in Table 2-21. Reconfiguration in the cloud is the fastest option, due to the use of the dedicated HBICAP that enables high-speed reconfiguration from bitstreams that come as DMAs transfers over the PCIe. The fog platform follows, benefiting from dedicated reconfiguration engines implemented in silicon [Xilinx'19b], while the edge platform exhibits the slowest reconfiguration. Nevertheless, the smaller logic capacity of edge devices results in reduced bitstream sizes, which partially compensates for the lower reconfiguration throughput. These differences highlight that the optimal platform

choice also depends on reconfiguration frequency and bitstream size.

Finally, Table 2-22 quantifies the overhead introduced by using the proposed infrastructure (i.e., running containerized applications and deploying them in the different continuum nodes with Kubernetes), compared to running them locally without any virtualization. The worst-case scenario resulted in a 1.23% increase in execution time compared to direct execution without virtualization. This minor overhead is well justified by the portability and scalability benefits provided by the proposed infrastructure.

Table 2-22: Execution overhead per platform on the cloud-edge continuum infrastructure.

Platform	Overhead	
	Mean	**Std. Deviation**
Cloud	0.90%	0.20%
Fog	0.41%	0.56%
Edge	1.23%	1.67%

Original contribution 2-5 *A dwarf-based strategy based on HLS benchmarks to characterize and validate the proposed infrastructure, including the monitoring framework, the node-level workload management (acceleration deployment and monitoring) and its extension to the cloud-edge continuum.*

2.5.4 Conclusions

This chapter introduces a workload management infrastructure designed to facilitate the efficient deployment and execution of dynamic hardware-acceleration workloads across the cloud-edge continuum. Leveraging an extended version of the ARTICo^3 framework with cloud-based FPGAs and multi-tenant support, the infrastructure provides mechanisms for managing and offloading workloads to the FPGA fabric. An integrated monitoring framework offers run-time power and performance trace collection. To further enhance scalability and portability, workloads are virtualized in containers and orchestrated through Kubernetes, while Liqo enables multi-cluster integration across heterogeneous nodes. Together, these components establish the foundation for seamless workload management, deployment, and monitoring in distributed heterogeneous reconfigurable systems.

The validation campaigns conducted in this chapter provide two key insights that support the contributions of this Thesis.

First, leveraging a heterogeneous set of devices, ranging from low-power edge platforms to high-performance cloud FPGAs, enables users to explore a big space

of trade-offs between performance, energy efficiency, and resource availability. This diversity enables application users to execute workloads on the most suitable hardware for their specific requirements, whether their needs are latency-sensitive processing at the edge or massively parallel acceleration in the cloud. This observation alone supports the value of the cloud-edge continuum as a viable computing paradigm.

Second, the results have shown that the proposed cloud-edge continuum infrastructure effectively supports FPGAs in this computing paradigm by enabling the seamless deployment of hardware acceleration workloads across all layers. The proposed solution has proven to scale well across nodes with vastly different capabilities. Additionally, the infrastructure introduces minimal run-time overhead, validating its practicality.

Moreover, the integrated monitoring framework complements this infrastructure by offering flexible, non-intrusive power consumption and performance monitoring across a diverse set of platforms. The framework adapts to the specific hardware and software constraints of each node, providing either high-resolution traces or lightweight measurements depending on the configuration. These capabilities are essential not only for debugging and monitoring, but also for enabling advanced self-aware or adaptive execution strategies, which will be explored in future chapters.

Chapter

3 DATA-DRIVEN MODELING OF DYNAMIC WORKLOADS

This chapter explores data-driven modeling strategies to characterize the dynamic workloads present in the cloud-edge continuum. The proposed modeling strategy aims to predict the power consumed by an FPGA device when accelerating a given workload and its execution time. The infrastructure presented in the previous chapter has been extended to support the run-time implementation of this modeling methodology with minimal overhead.

First, an overview of the components that compose the proposed modeling approach is presented, followed by a section describing a synthetic workload generator used to build real-world dynamic workloads for validation purposes. A subsequent section exhaustively analyzes the interaction between kernels when managing dynamic workloads, which motivates the need for data-driven modeling in the first place. Then, state-of-the-art data-driven modeling methodologies are studied. The chapter continues by describing the modeling strategy proposed in this Thesis, providing details about its implementation: first as a static approach for predefined workloads and then as an incremental extension for run-time workload characterization. Finally, a set of validation tests evaluates the proposal.

3.1 Chapter Overview

In the conventional reconfigurable system implementation scenario where the hardware accelerators of the workload are fixed and known at design time, a Design Space Exploration (DSE) process could determine, before deployment, the number of replicas of each accelerator and when they must be reconfigured and executed in the FPGA fabric, to optimize metrics such as throughput or power consumption. However, such an approach cannot be implemented in a cloud-edge continuum scenario where the system faces dynamic workloads that might change at run time. Therefore, to efficiently schedule, deploy and execute these dynamic workloads and keep the system working at its optimal operating point in terms of Quality of Service (QoS) or power consumption constraints, resource management infrastructures like the one presented in Chapter 2 need a run-time workload characterization mechanism that provides accurate information about the kernels that need to be executed, as well as extracts system-level details (e.g., resource contention).

For example, power consumption is a major challenge in cloud-edge continuum scenarios: as the computation moves closer to the edge devices, the energy budget becomes more restricted [Shi'16]. Accurate power consumption characterization might enable proactive energy management techniques that would result in more energy-efficient deployments. Similarly, performance characterization might also be crucial, for instance, in real-time applications, where this information can be leveraged to make optimal task scheduling decisions to meet requirements.

However, this characterization process cannot be achieved by just modeling each kernel in isolation since that approach does not consider aspects (e.g., resource contention) that appear due to the interaction between kernels, which is present in scenarios with multiple accelerators simultaneously executed in the FPGA, like the one targeted in this Thesis. Likewise, the number of variables involved in such a complex scenario makes constructing analytical models unfeasible.

In contrast, this Thesis proposes a data-driven modeling methodology based on ML algorithms, since they are particularly good at finding these complex relationships between multiple factors. The proposed modeling methodology leverages power consumption and performance traces gathered with the monitoring framework presented in Chapter 2 in a way that, after the workload has been executed and monitoring data has been collected, ML-based models are trained to predict the power consumption and the performance of reconfigurable multi-accelerator systems.

Nevertheless, while such an approach might suffice for static scenarios, the dynamic nature of the workloads present in the cloud-edge continuum can produce unexpected changes in the workload (e.g., run-time requirements, resource availability, environmental/system variations) that introduce challenges that traditional, offline models cannot effectively deal with. Therefore, to obtain an optimal power consumption and performance characterization, predictive models need to be able to adapt to these dynamic conditions [Hadsell'20], in order to learn continuously and deal with model drift.[1] As a result, the baseline modeling methodology has been extended to an incremental ML implementation, enabling the models to be dynamically updated and adapt to the changing conditions of the cloud-edge continuum. Moreover, this approach has been tested on a diverse set of platforms, proving to be platform-agnostic and portable across the nodes of the continuum.

Figure 3-1 presents the main components of the incremental modeling methodology highlighted with respect to the rest of this Thesis's components. The modeling process starts by receiving power consumption and performance traces from the monitoring framework, which are collected at run time during the execution of dynamic workloads. The extracted traces are then processed at run time to generate meaningful observations. Subsequently, these observations are fed to the ML-based models to train them.

Incremental learning is typically performed by continuously training the models on new observations. However, that approach could be too resource-hungry, specifically

[1] Model drift refers to the degradation of the model due to changes in the environment [Bayram'22]. In this case, it could correspond to changes in the workload or the state of the nodes in the continuum.

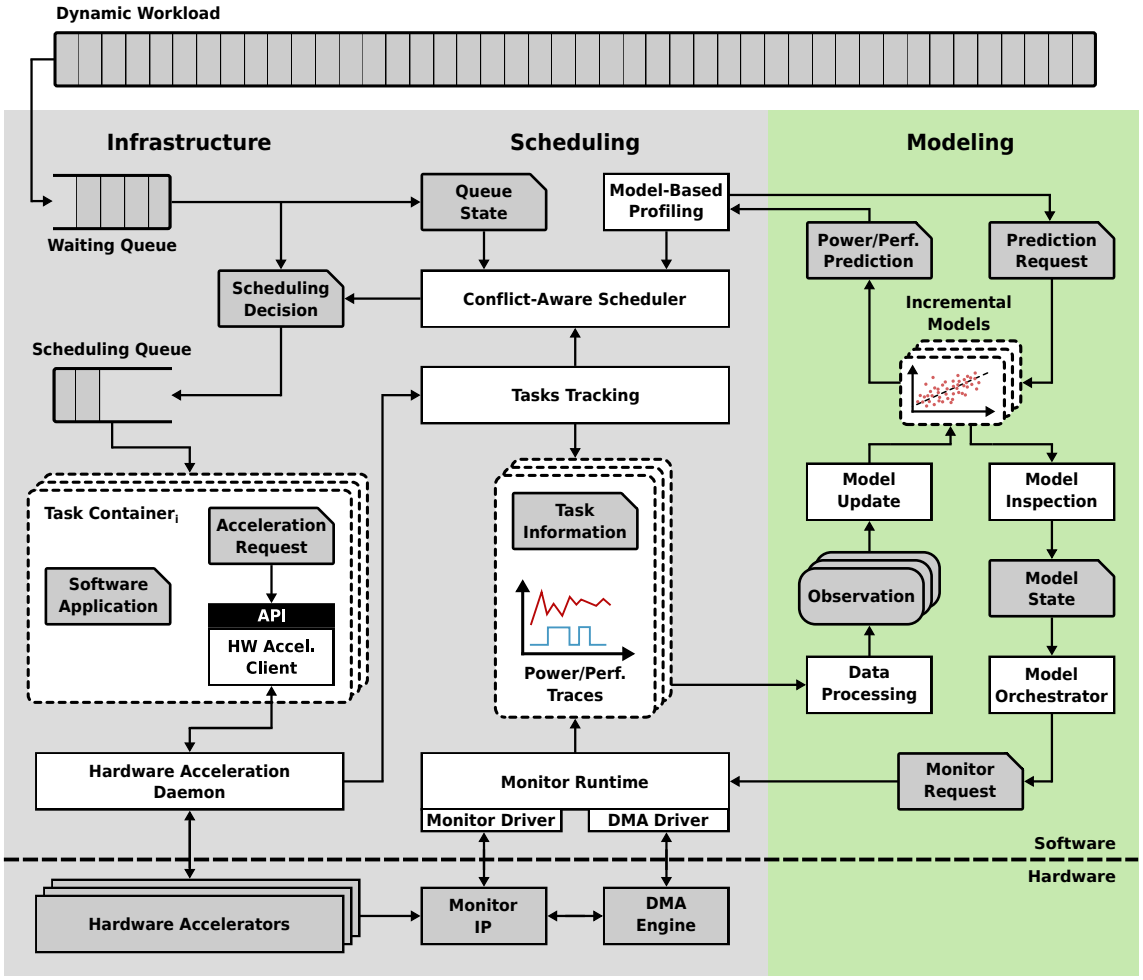


Figure 3-1: Details of the modeling methodology proposed in this Thesis.

in low-end devices such as those present at the edge. To tackle this, a resource-aware learning mechanism has been designed. It operates by i) inspecting the models to gather their state (e.g., prediction error) and ii) deciding whether to keep updating the models (by requesting more traces from the monitoring framework). Its goal is to keep the models up-to-date while minimizing training time and, therefore, modeling overhead.

This whole process is repeated forever, ruled by the model orchestrator mechanism, updating the models incrementally at run time. Please note that the incremental models obtained with this modeling methodology are then leveraged in Chapter 4 to perform run-time task scheduling decisions.

The incremental modeling methodology is open source and available online.[2]

[2]https://github.com/des-cei/fpga-modeling/

3.2 Synthetic Workload Generator

As has been described before, a significant challenge when dealing with a cloud-edge continuum scenario is the dynamic nature of its workloads.[3] Therefore, having access to a realistic dynamic workload is a major requirement to produce a fair assessment of the components presented in this Thesis. Moreover, focusing specifically on the modeling aspect that affects this section, the extraction of power consumption and performance traces (with the monitoring framework) must be under a diverse set of circumstances to ensure that the models that will be implemented not only capture the specific characteristics of the workload being executed but also can be generalizable, even when dealing with unforeseeable conditions. To satisfy this need, a dynamic workload generator has been designed, capable of creating diverse workload patterns by tuning the following configuration parameters:

- **Number hardware acceleration requests (N):** refers to the number of hardware acceleration tasks that form the workload. Each task corresponds to a specific kernel function.
- **Mean inter-arrival time ($\overline{T_I}$):** represents the mean time gap between two consecutive hardware acceleration tasks.
- **Kernel function pool (K_P):** denotes the collection of kernel functions available for execution. Each request selects one kernel function from this set.
- **Job size range (J_R):** indicates the range of possible job sizes (i.e., the number of times a kernel has to run per request) that can be assigned to a request.
- **Accelerator replica range (A_R):** specifies the upper limit on the number of simultaneous replicas of a kernel that may execute concurrently in an SIMD-style configuration.

The workload generator produces a synthetic dynamic workload that consists of N hardware acceleration requests. The i^{th} request is represented by an arrival time ta_i, and a kernel function $k_i \in K_P$ with job size $j_i \in \{0, J_R\}$ and $a_i \in \{0, A_R\}$ accelerator replicas running in parallel. k_i, j_i, and a_i are selected from within their respective ranges following a uniform distribution, ensuring that each kernel function, job size, and number of accelerator replicas are represented equally in the synthetic workload. In contrast, the arrival time ta_i is constructed as the arrival time of the previous request ta_{i-1} plus an inter-arrival time sampled from a Poisson distribution, with a mean inter-arrival time of $\overline{T_I}$. Please note that the number of events (hardware acceleration requests, in this case) that occur within a fixed time interval is typically assumed to follow a Poisson distribution [Kingman'92], as observed in datacenters [Kandula'09].

[3] Please refer to Section 1.4.2.

An example of this synthetic workload generation process is represented in Figure 3-2, where each hardware acceleration request of the dynamic workload is generated one by one. The presented example highlights the parameter creation of a workload request: 42 ms of arrival time, three accelerator replicas in parallel, 2048 of job size, and a Type B kernel function. Each of the n hardware acceleration requests that compose the generated workload is created following this very same process. Please note that a particular set of parameters represents each request.

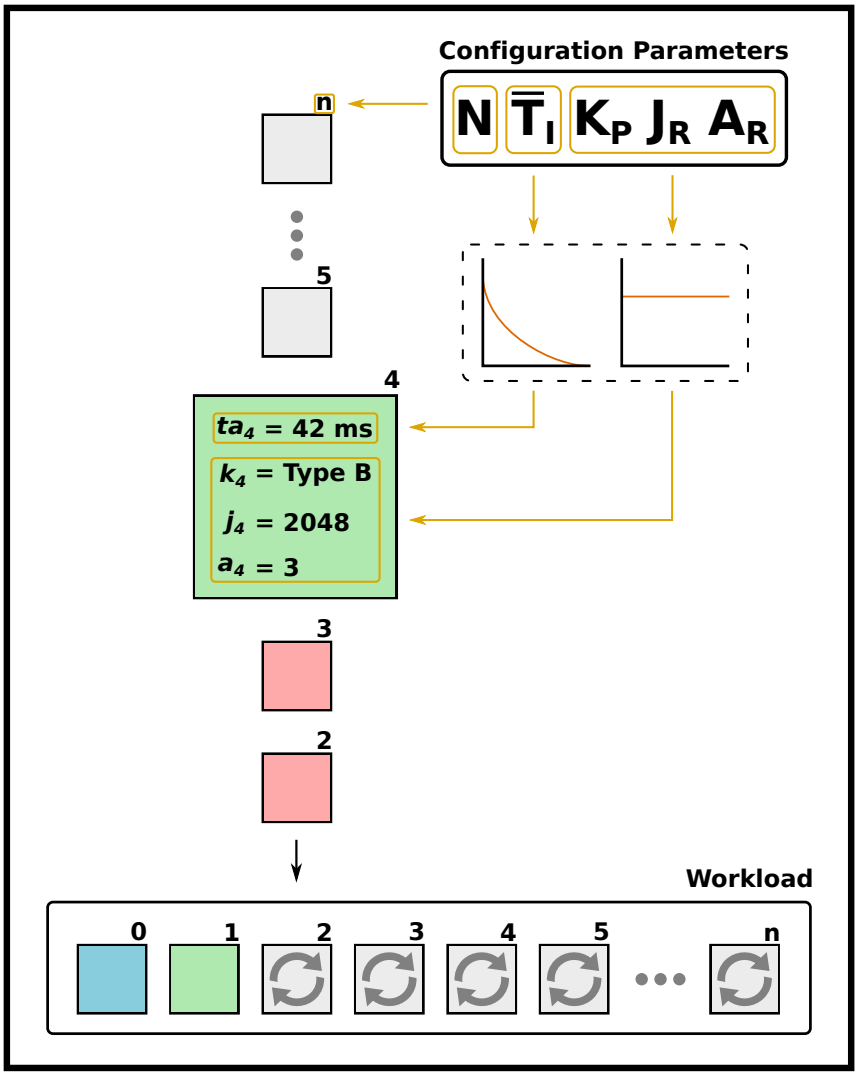


Figure 3-2: Dynamic workload generation process. The bottom queue represents each of the n hardware acceleration requests composing the synthetic workload being generated. The color of each request refers to its kernel function type, where the gray loading symbol indicates that the parameters of that request have not been defined yet.

Every dynamic workload used for any validation campaign or similar in the remainder of this Thesis has been crafted with this synthetic workload generator. In any case, it will be indicated when employed.

3.3 Analysis of Kernel Interaction

Modeling reconfigurable systems faces a significant challenge in the interaction that occurs when running multiple parallel accelerators. Indeed, when dealing with the dynamic workloads present in the cloud-edge continuum, the modeling complexity increases even more, since different resource contention flavors (e.g., shared memory accesses, on-chip buses utilization) have to be taken into account in the process. To evaluate the significance of this interaction between kernels, a series of preliminary experiments has been conducted in a Zynq UltraScale+ device that integrates software and hardware processing elements (i.e., PS and PL, respectively).[4] For a more in-depth description of the setup used during these preliminary experiments, see Section 3.7.1.

The experiments that will be described next have been conducted by running workloads generated with the synthetic workload generator presented in the previous section. These workloads contain specific combinations of the MachSuite kernels described in Section 2.5.1. To execute these workloads, the workload management infrastructure presented in Chapter 2 has been integrated. Thanks to the monitoring capabilities embedded in the infrastructure, power consumption and performance traces can be extracted during the execution of the workloads. This section analyzes those traces first in a single-kernel scenario and then in a multi-kernel scenario. Please note that all experiments provide power consumption measurements from both the PS and the PL.

3.3.1 Single-Kernel Scenario

For this first analysis, the kernels present in the workload have been executed one by one, with no other kernel executed in parallel (i.e., without kernel interaction). Each kernel has been executed with various parallel accelerators (i.e., replicas).

Figure 3-3, Figure 3-4, and Figure 3-5 show, respectively, how the PS power consumption, the PL power consumption, and the execution time vary based on the executed kernel and the number of parallel accelerators configured. Please note that the data represented in these figures is calculated as the average of a set of observations. The specific number of observations representing each kernel varies, since the monitoring framework collects measurements periodically, causing the probability of capturing a kernel execution to be dependent on its execution time (long-lasting kernels generate more observations). In any case, each kernel is represented with a number of observations ranging from 20 to 800.

One observes that the PS power consumption, in Figure 3-3, decreases with the number of parallel accelerators. The reason behind this counterintuitive behavior is the way ARTICo^3 transfers data to the accelerators. Since ARTICo^3 has only one port for data transfers, executing multiple accelerators in parallel produces resource

[4] XCZU9EGFFVB1156-2-I device.

contentions (bus and DMA) that slightly increase their execution time. This increase in execution time effectively reduces the duty cycle of the PS, which translates to a reduction of the average PS power consumption as the number of parallel accelerators increases. Nevertheless, since the power consumed by the processor is mainly related to OS procedures independent of the hardware accelerators being executed, the variation in PS power consumption is insignificant. This exact same reason explains why the type of kernel executed does not alter the PS power consumption.

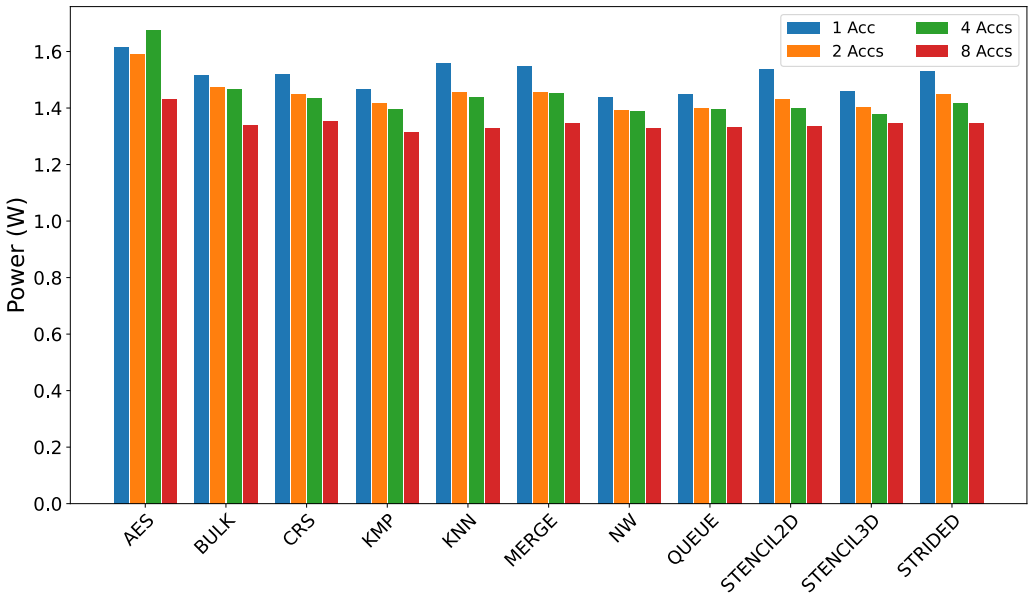


Figure 3-3: Measured average PS power consumption according to kernel type and number of accelerators. Data processed (job size) scales linearly with the number of accelerators.

Conversely, the PL power consumption, see Figure 3-4, increases with the number of parallel accelerators. Though intuitive this time, it is noteworthy how slight the variation in power consumption is. Moreover, the baseline power consumption (i.e., one accelerator scenario) is practically kernel-independent. The reason is the high static power consumption inherent to SRAM-based FPGA devices, which makes the dynamic power consumption less apparent, particularly due to the low-level reconfiguration constraints (i.e., equivalent reconfigurable region size regardless of the resource utilization of the different kernels). However, there are still specific kernels (e.g., KNN, STRIDED) that make the dynamic power consumption more apparent, as the multi-accelerator scenarios indicate.

Finally, the execution time, shown in Figure 3-5, presents a significant variation due to the type of kernel or the number of parallel accelerators executed. In turn, regarding the execution time variation due to the number of accelerators, it is essential to mention that the execution time metric presented in the experiments measures the time it takes to execute n accelerators one time. Thus, the data processed does not remain constant; the job size increases proportionally to the number of parallel accelerators. Again, due to the single port for data transfers of ARTICo^3, the more

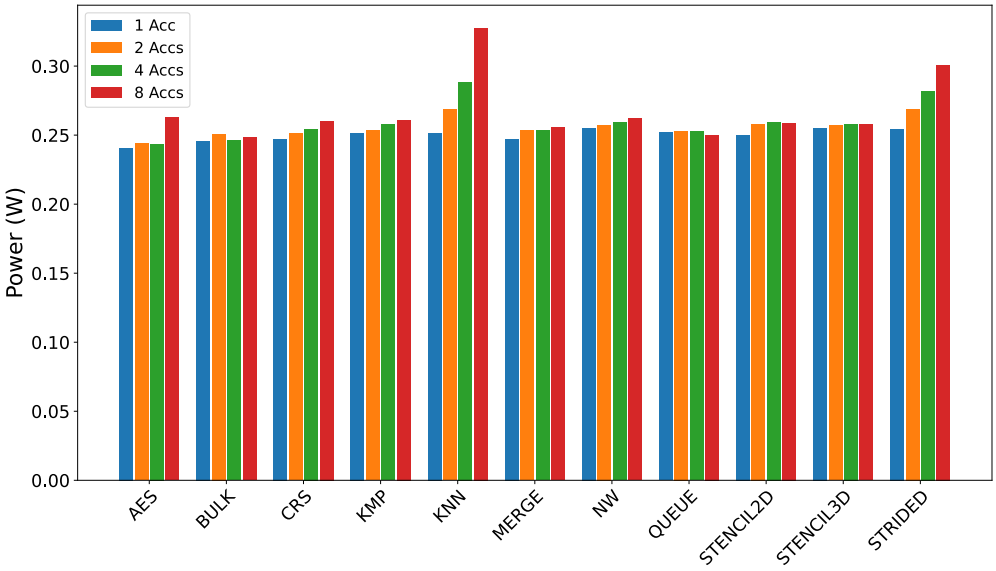


Figure 3-4: Measured average PL power consumption according to kernel type and number of accelerators. Data processed (job size) scales linearly with the number of accelerators.

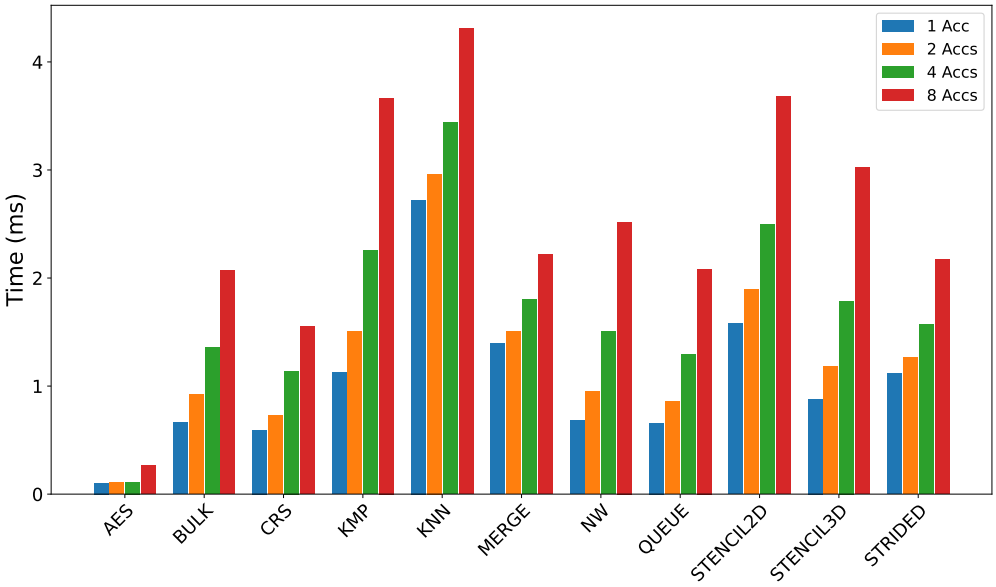


Figure 3-5: Measured execution time according to kernel type and number of accelerators. Data processed (job size) scales linearly with the number of accelerators.

accelerators configured in parallel, the more contention appears. This phenomenon explains the growth in execution time due to the number of accelerators. However, if normalized with the job size, the execution time decreases as the number of accelerators increases, as depicted in Figure 3-6.

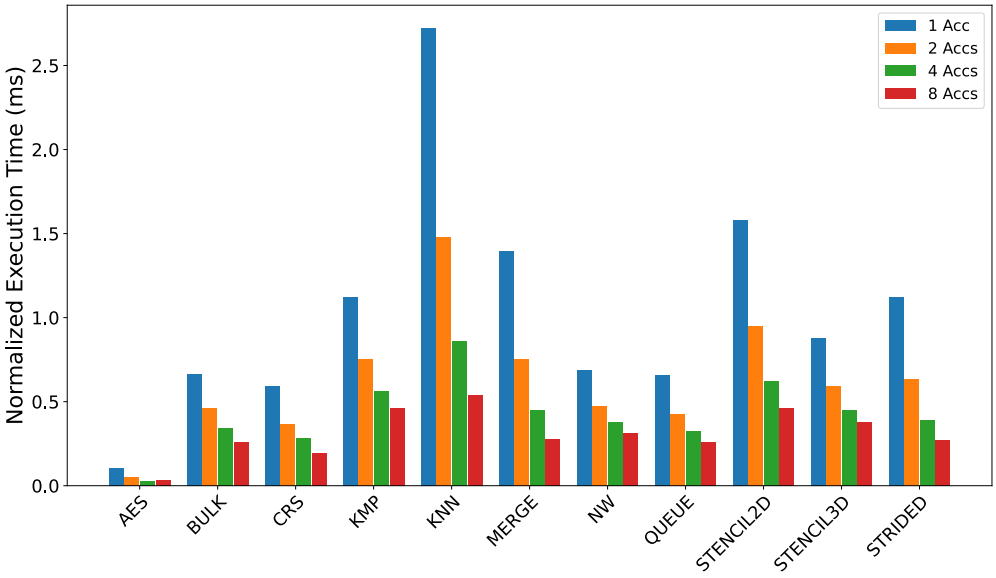


Figure 3-6: Normalized execution time according to kernel type and number of accelerators. Data processed (job size) is kept constant, regardless of the number of accelerators.

While this section analyzed how the system behaves in scenarios with a single kernel, the cloud-edge continuum faces workloads where multiple kernels have to be simultaneously executed in parallel, interacting with each other and affecting the power consumption and performance of the system. That multi-kernel interaction will be evaluated next.

3.3.2 Multi-Kernel Scenario

For this second analysis, the kernels present in the workload have been executed in an FCFS fashion, allowing multiple kernels to be executed in parallel (i.e., with kernel interaction). Each kernel is executed with various parallel accelerators (i.e., replicas).

The number of possible kernel combinations (each kernel can present a variable number of parallel accelerators) is so large that creating a valuable graphical representation of them to analyze the interaction between multiple kernels is not feasible due to the high-dimensional space.

As a first and broader approach to illustrate the effects of kernel interaction, Figure 3-7, Figure 3-8, and Figure 3-9 have been developed. These figures represent, respectively, the PS power consumption, PL power consumption, and execution time of a kernel when executed alongside an arbitrary number of parallel accelerators (of any other kernel type, including itself), normalized to the value observed when the kernel is executed as a single accelerator in isolation (i.e., when no kernel interaction nor resource contention is present).

Each bar in the figures represents a statistical distribution, illustrating the impact of kernel interaction on a specific kernel when executed in parallel with a varying number

of accelerators (of any kernel type), from isolated execution (leftmost bar) to scenarios involving eight parallel accelerators (rightmost bar). These distributions are built by grouping all the observations for each number of parallel accelerators and normalizing the magnitude under evaluation to the value of that magnitude corresponding to the isolated execution of that very same kernel with a single accelerator replica. The average and standard deviation are then computed for each distribution, representing the impact and variability introduced by the kernel interaction.

Focusing on Figure 3-7, the bars represent statistical distributions of normalized PS power consumption for each number of parallel accelerators:

- The first bar serves as the baseline distribution. It includes all observations in which only a single accelerator (of any kernel type) is executed. Each observation is normalized to itself, resulting in a mean of 1 and a standard deviation of 0. This distribution is a reference point for subsequent comparisons.
- The second bar corresponds to the distribution formed by all observations involving exactly two parallel accelerators, either two replicas of the same kernel or two distinct kernels. For each kernel under evaluation, its PS power consumption is normalized to the mean PS power consumption of that very same kernel in the isolated execution case (from the first bar). The resulting distribution represents how the presence of an additional accelerator affects the PS power consumption of the kernel under evaluation.
- Subsequent bars (up to seven additional parallel accelerators) follow the same approach. Observations are grouped according to the number of parallel accelerators, forming distributions, and the PS power consumption of each kernel under evaluation is normalized to its isolated baseline. Let us imagine an example where two AES, two KNN, and four STRIDED accelerators are executed simultaneously (a total of eight). In such a scenario, for each kernel under evaluation in each observation, the PS power consumption normalized to its isolated execution (e.g., being AES the kernel under evaluation) would be represented in the rightmost bar, since there is a kernel under evaluation (i.e., AES) and seven additional parallel accelerators.

Figure 3-8 and Figure 3-9 follow the same methodology to depict, respectively, the statistical distributions of PL power consumption and execution time under different degrees of kernel interaction. These representations enable a comprehensive analysis of the impact produced by the kernel interaction.

Moving across the horizontal axis in these figures, one sees the effect of kernel interaction in the magnitude under study (i.e., PS power, PL power, execution time) when the number of additional parallel accelerators changes. The blue bars, representing the mean of each distribution, portray the tendency of the magnitude under study based on the number of additional parallel accelerators. The black lines, representing the population that falls within one standard deviation from the mean

(i.e., ≈ 68%), indicate the variability of the magnitude under study depending on the number of additional parallel accelerators.

Regarding the PS power consumption, Figure 3-7 shows a slight reduction in the mean of the normalized power consumption as the number of additional parallel accelerators increases, a phenomenon consistent with the analysis carried out previously for the single-kernel scenario. Note also a slight variability (represented with the standard deviation) in the power consumption, following the same trend.

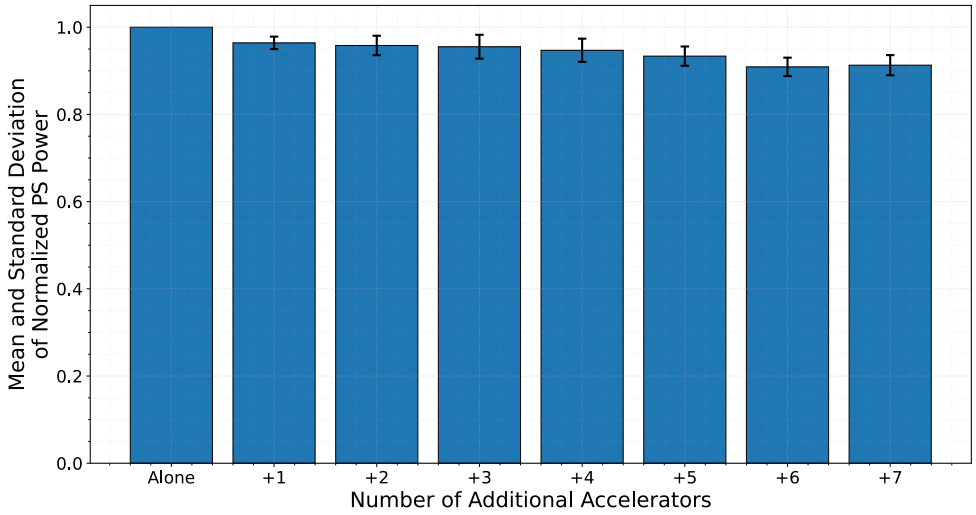


Figure 3-7: Variability in PS power consumption due to the kernel interaction. Each bar represents the statistical distribution of the normalized PS power consumed by a kernel when executed in parallel with other accelerators.

When analyzing the PL power consumption in Figure 3-8, the tendency again follows the assessment made for the single-kernel scenario. There is a slight increase in the mean of the normalized power consumption as the number of additional parallel accelerators increases. In turn, the variability of the magnitude, although higher than the PS case, remains not substantial.

The execution time (see Figure 3-9), contrary to the other magnitudes under study, is significantly affected by the kernel interaction. Observe that the mean normalized execution time rises up to 3x when seven additional accelerators are executed in parallel. Moreover, the magnitude variability also scales with the number of additional parallel accelerators.

These figures summarize how significant the impact of kernel interaction in the magnitude under study is as the number of additional parallel accelerators increases (execution time is particularly affected).

To analyze the impact of kernel interaction more deeply, a two-kernel scenario, a simplified variant of the already studied multi-kernel scenario, will be explored, since it eases the graphical representation. This representation has been conducted by generating a series of interaction maps. The interaction maps for PS power consumption, PL

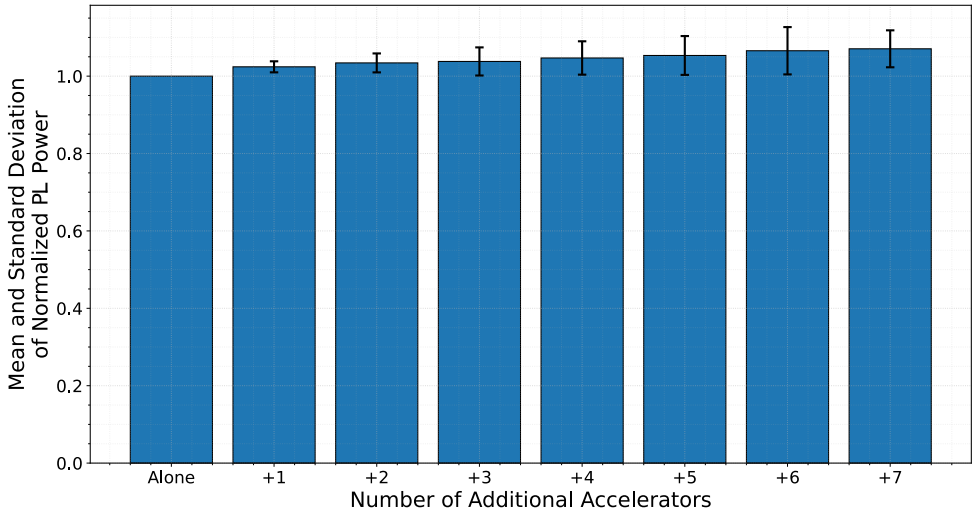


Figure 3-8: Variability in PL power consumption due to kernel interaction. Each bar represents the statistical distribution of the normalized PL power consumed by a kernel when executed in parallel with other accelerators.

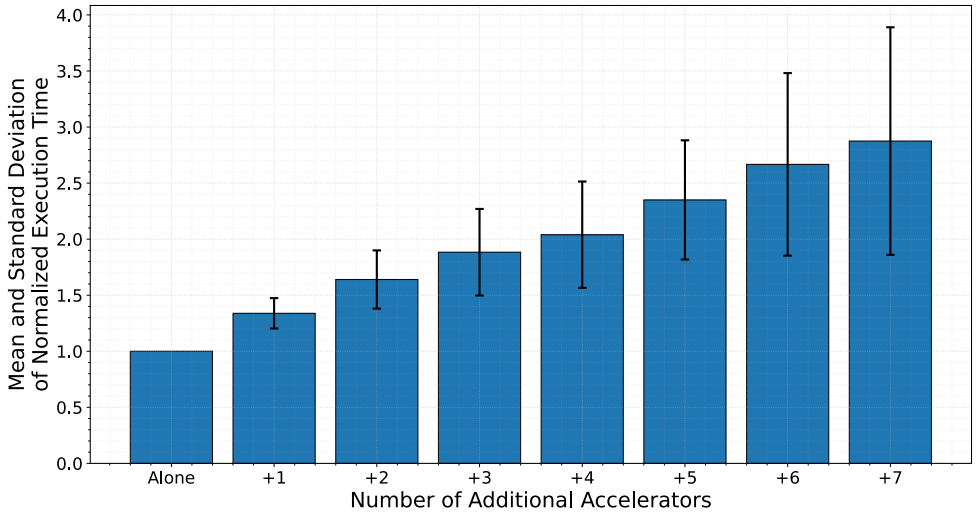


Figure 3-9: Variability in execution time due to kernel interaction. Each bar represents the statistical distribution of the normalized execution time of a kernel when executed in parallel with other accelerators.

power consumption and execution time are represented, respectively, in Figure 3-10, Figure 3-11 and Figure 3-12.

Each cell of these interaction maps represents a specific set of two kernels executed alongside in the FPGA fabric. The rows (i.e., Y axis) dictate which is the kernel under evaluation, while the columns (i.e., X axis) indicate which is the second kernel that executes in parallel with the one under evaluation (i.e., the kernel that is, therefore, interacting with the kernel under evaluation). This simplified scenario evaluates only the execution of two kernels in parallel, each configured with two accelerator replicas

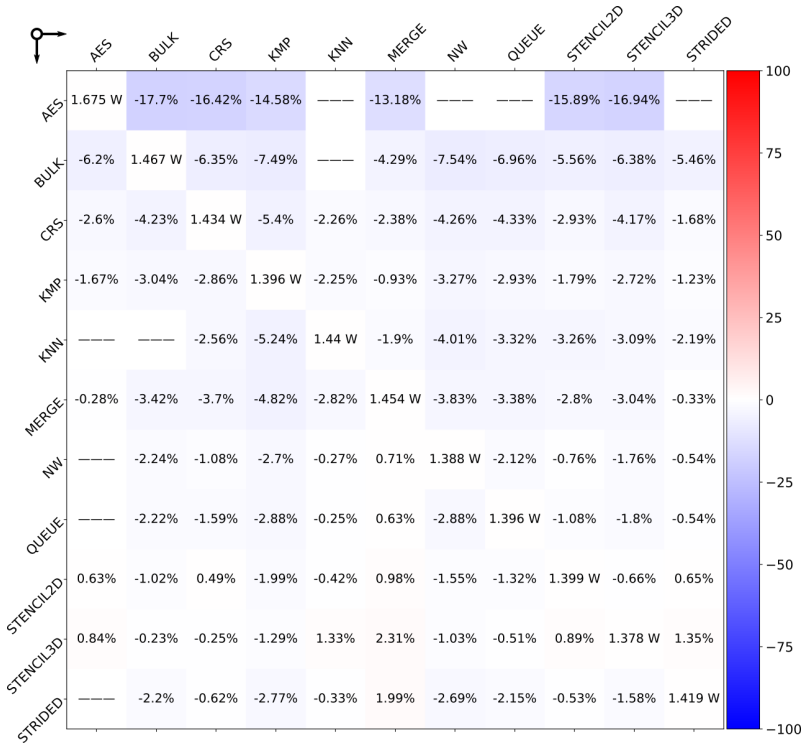


Figure 3-10: Measured kernel interaction impact on PS power consumption (y-axis: kernel under evaluation, x-axis: kernel executed in parallel to it).

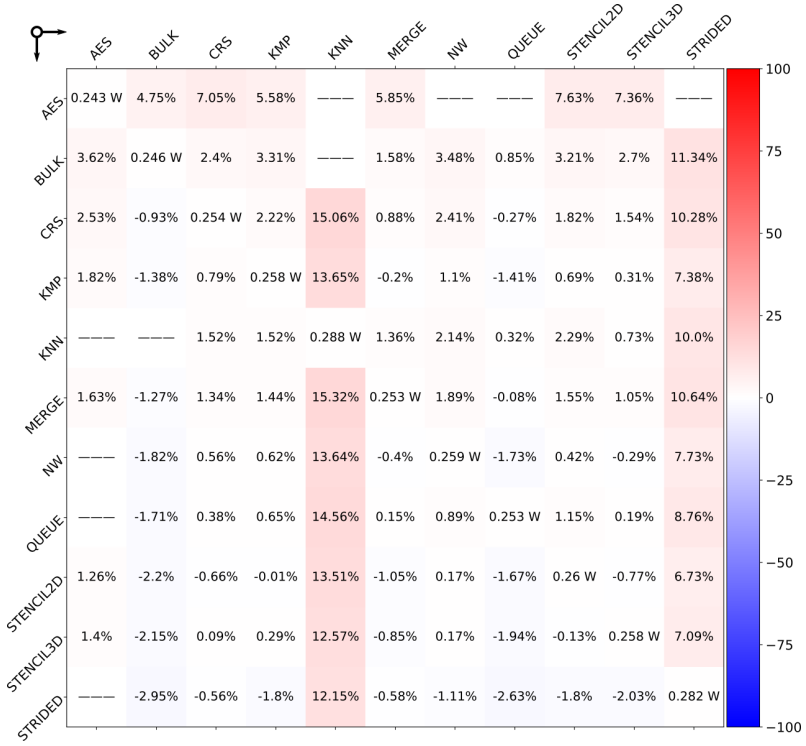


Figure 3-11: Measured kernel interaction impact on PL power consumption (y-axis: kernel under evaluation, x-axis: kernel executed in parallel to it).

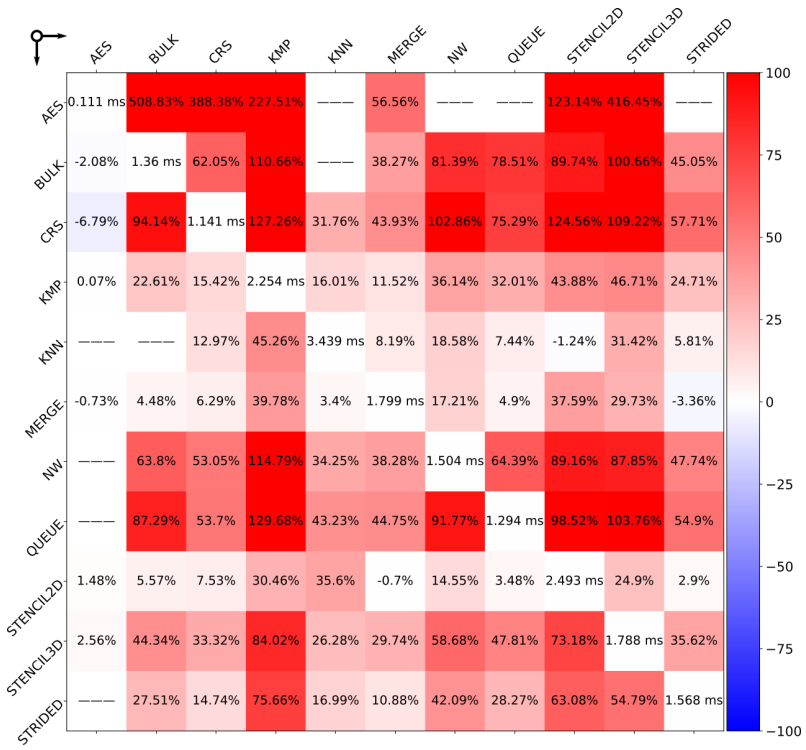


Figure 3-12: Measured kernel interaction impact on execution time (y-axis: kernel under evaluation, x-axis: kernel executed in parallel to it).

(for a total of four accelerators). On the diagonal of the interaction maps, each cell represents the execution of two accelerators of a single kernel, the kernel under evaluation, without any kernel interaction.[5] That is why diagonal elements are treated as the reference magnitude (either power consumption or execution time) for each corresponding kernel under evaluation.

Let us see an example for the sake of clarity. Focusing on the interaction map of the execution time shown in Figure 3-12, the second row, second column cell, which is on the diagonal, represents the execution time of the BULK kernel when executed alone. In contrast, any other cell outside the diagonal but in the same row represents the relative difference between the execution time of BULK (i.e., the kernel under evaluation, represented by the row) when executed in parallel with any other kernel (represented by the column), and the execution time of BULK when executed alone. For instance, in Figure 3-12, the second row, third column cell indicates that executing BULK in parallel with CRS increases the time it takes to execute BULK by 62%, compared to only executing BULK alone. To emphasize the significance of the effect of kernel interaction, every cell has been given a color: red means the magnitude under study increases, while blue means the opposite. The darker the color, the greater the impact of kernel interaction. The null values present in some cells indicate that

[5]Please note that, for the execution time, single kernels (i.e., those located in the diagonal) are executed with two replicas; meanwhile, for the PS and PL power consumption graphs, single kernels have been executed with four replicas, to have a fair comparison with two-kernel cells.

no traces have been measured for that particular combination of kernels during the experimental campaign. Note that the magnitude specified in each cell represents the average of a set of measured observations, the size of which is dependent on the specific kernel combination, as mentioned in the single-kernel scenario.

The data shown in Figure 3-10 is consistent with the analysis of the single-kernel scenario. The kernel interaction translates into a reduction in the PS power consumption, but its impact is not significant.

When focusing on the PL power consumption shown in Figure 3-11, the data remain consistent with the findings discussed in the single-kernel scenario. The PL power consumption increases due to kernel interaction, but it is not significantly affected. However, certain combinations of kernels where the kernel under evaluation is executed in parallel with a high power-demanding kernel show an increase in PL power consumption of 10%-15%. This, again, is explained by the higher dynamic power consumption of those kernels, which considerably increases the overall PL power consumption, regardless of the kernel under evaluation.

Regarding execution time, the data represented in Figure 3-12 shows that the execution time of any kernel is strongly affected by the interaction with the others. It can be seen that specific kernels, particularly the fastest ones like AES or CRS (focus on their rows), are strongly affected by the kernel interaction with slowdowns of up to 5x (i.e., 500% execution time variation). At the same time, they have a minor impact on the other kernels (focus on their column). Conversely, the slowest kernels, such as KMP or STENCIL2D, are the kernels least affected by kernel interaction, while impacting the other kernels the most. The reason is simple: when a kernel execution is short, the overhead due to resource contention has a higher relative impact on it. Similarly, its effect on other kernels is minimal since its execution (and the contention it can cause) is small. Kernels with longer execution show the opposite behavior.

The analysis of this section demonstrates that kernel interaction exists and indeed has a significant and intricate impact on the behavior of the system. This kernel interaction complexity justifies exploring the application of data-driven modeling methodologies to characterize the execution of dynamic workloads on reconfigurable multi-accelerator systems in the cloud-edge continuum scenario.

Original contribution 3-1 *An evaluation of the impact on power consumption and computing performance of the interaction between kernels when executing dynamic workloads in reconfigurable multi-accelerator systems.*

3.4 State of the Art

The analysis of the state of the art in this chapter is divided into distinct sections. First, the works in literature that explore the use of traditional, offline ML to model computing platforms are analyzed. Then, state-of-the-art characterizations of dynamic workloads using incremental learning are studied.

3.4.1 Offline Modeling

ML methodologies have become increasingly popular in the characterization and modeling of computing platforms, particularly due to their capacity to manage the complexity and variability of modern workloads. These methods have been adopted across a range of computing architectures, including CPUs, GPUs and FPGAs, offering an alternative to traditional analytical models. A comprehensive overview of ML implementations for workload modeling in these types of computing architectures can be found in [O'Neal'18], with a significant effort focused on heterogeneous multiprocessor platforms [Pagani'20].

An example is the work presented in [Li'17], which introduces a thread characterization method that takes advantage of hardware performance counters together with ML models. This approach estimates workload execution using software-derived metrics such as instructions per cycle and cache miss rates. A Principal Component Analysis (PCA) is employed for dimensionality reduction. The resulting reduced feature set is then processed with Support Vector Regression (SVR) and Linear Regression (LR) models. The trained models are capable of predicting the temporal evolution of threads, enabling the identification of distinct execution phases, such as memory-bound and compute-bound stages.

Neural Network (NN) have also demonstrated considerable effectiveness in this domain. Specifically, in the context of many-core systems featuring Static Non-Uniform Cache Architectures (S-NUCAs), NNs have achieved greater accuracy and lower overheads when compared to analytical approaches [Rapp'21]. These findings highlight the strengths of data-driven techniques in modeling complex systems.

Proactive power management represents another key application area for ML in system modeling. In [Gupta'20], the authors propose a method based on LR to rule a Dynamic Voltage and Frequency Scaling (DVFS) procedure in multi-core architectures. The model is trained using a set of hardware performance counters and architectural parameters, such as core frequency and instructions per cycle, to restrict power consumption without compromising performance. A complementary strategy is introduced in [Tian'21], where deep Reinforcement Learning (RL) is used to guide the DVFS mechanism. In this approach, the learner selects optimal voltage and frequency levels for individual cores based on their computational intensity, thus increasing energy efficiency.

In addition, ML techniques have also been widely applied to GPUs. In [Baldini'14], supervised learning models are trained, using dynamic profile data collected from the execution of code on general-purpose processors, to predict GPU performance. These models aim to predict the most suitable computing fabric for a function by solving a binary classification problem. Another significant contribution is made in [Amarís'16], where the authors compare LR, Support Vector Machines (SVMs), and Random Forests (RFs) with an ad-hoc analytical model for GPU execution time estimation. The study concludes that although analytical models can outperform ML methods, they often require manual inclusion of complex low-level details of the interaction between the architecture components. In contrast, the ML models demonstrate stronger generalization capabilities across workloads and architectures.

Concerning reconfigurable systems, particularly FPGAs, ML has predominantly been used to optimize the DSE process [O'Neal'18]. Many studies have proposed data-driven models to estimate performance and resource utilization in the early stages of the DSE, reducing the need for exhaustive simulation or synthesis phases. For instance, in [Lopes'20], the authors develop ML models capable of predicting performance across different configurations of integrated CPUs and CGRAs, enabling faster exploration of micro-architectural variants without requiring their actual implementation. Similarly, [Li'22] presents a methodology for fast resource estimation of RTL designs. The approach involves extracting a set of descriptive features from RTL code, which are then used as inputs to train ML algorithms to predict the required FPGA resources.

The Pyramid framework introduced in [M. Makrani'19] extends these ideas to HLS, using ML to predict resource usage and execution performance accurately. Approaches like this can help designers minimize the need to manually specify optimization directives, simplifying the HLS workflow. An alternative approach is demonstrated in [Goswami'22], where the authors propose a predictive tool that eliminates the need for prior synthesis relying on features derived from high-level code, the LLVM intermediate representation, and the control- and data-flow graphs of the design. A regression model trained on these characteristics predicts performance and resource utilization metrics.

ML has also been used in the context of DPR for FPGA systems. For instance, [Pham'16] proposes a framework that leverages the ML models to minimize the reconfiguration overhead. The approach covers the scheduling, placement, and post-placement phases. Once a set of Pareto-optimal configurations is generated using Genetic Algorithm (GA), clustering and regression techniques are trained on task graph datasets to use them later for inferring Pareto-optimal design points.

Unlike state-of-the-art works, the proposal of this Thesis follows a run-time data acquisition approach, performing a workload characterization rather than a DSE analysis at design time. Furthermore, this work targets the domain of high-performance embedded applications deployed in FPGA-based systems. In addition, the proposed methodology is meant to be implemented in any layer of the cloud-edge continuum, rather than focusing on the cloud like most works [Hormozi'12]. A

qualitative comparison between the works found in the literature and the proposal of this Thesis is presented in Table 3-1.

Table 3-1: State of the Art - ML for characterization and modeling of computing platforms.

Reference	Platform	Objective	ML Technique	Data Acquisition
[Li'17]	CPU	Perf.	PCA + SVR/LR	Run-Time
[Rapp'21]	Many-core	Perf.	NN	Run-Time
[Gupta'20]	Multi-core	DVFS	LR	Run-Time
[Tian'21]	Multi-core	DVFS	Deep RL	Run-Time
[Baldini'14]	CPU/GPU	Plat.	SVM	Run-Time
[Amarís'16]	GPU	Perf.	LR/SVR/RF	Run-Time
[Lopes'20]	CPU + CGRA	Perf. (DSE)	Multiple Reg.	Design-Time
[Li'22]	FPGA (RTL)	Rsc. (DSE)	Multiple Reg.	Design-Time
[M. Makrani'19]	FPGA (HLS)	Rsc. + Perf. (DSE)	Multiple Reg.	Design-Time
[Goswami'22]	FPGA (HLS)	Rsc. + Perf. (DSE)	Multiple Reg.	Design-Time
[Pham'16]	FPGA (HLS)	Reconfig. (DSE)	GA + Cluster + LR	Design-Time
This work	FPGA	Power + Perf.	SVR/RT	Run-Time

AND (+) | **OR** (/)

Objective
Perf.: Performance estimation
Plat.: Platform selection
Power: Power estimation
Reconfig.: Reconfiguration optimization
Rsc.: Resource estimation

ML Technique
Multiple Reg.: Evaluation of multiple regression algorithms (e.g., LR, SVR, NN)

3.4.2 Incremental Learning

The approaches shown in the section above are meant to be used in static environments where workloads are already known at design time and can be characterized beforehand. However, in environments where workloads and system conditions evolve, such as the cloud-edge continuum, the ability to have real-time awareness and adapt accordingly is essential. This requirement highlights a fundamental limitation of static workload characterization. In turn, incremental learning addresses this limitation by continuously updating predictive models throughout the system's lifetime, thereby mitigating the degradation in model accuracy caused by model drift or unexpected environmental variations during run time, which effectively enables run-time self-adaptation [Giraud-Carrier'00, He'20].

The primary objectives of incremental learning in dynamic systems are twofold. First, predictive models must remain accurate even when the data distribution or relationships among input variables change over time, a common phenomenon

in real-world scenarios [Bayram'22]. Second, models must remain updated while avoiding catastrophic forgetting, an issue in which newly acquired knowledge overrides information previously learned by the model [Chen'18]. A successful balance of adaptation and memory retention is a key challenge of incremental learning.

Most of the existing literature on incremental learning applied to workload characterization has focused on High-Performance Computing (HPC) systems, which offer abundant computational resources. These platforms can accommodate complex ML algorithms, such as Deep Neural Network (DNN), with competitive overheads. For instance, the framework proposed in [Tian'23] employs a static NN to predict the power consumption of HPC workloads. During run time, predictions are continuously compared against actual measurements. If the error exceeds a predefined threshold, the model is retrained on the latest data to update it. The results show that this incremental approach effectively counters model drift and improves predictive accuracy compared to purely static modeling methodologies.

Another notable example is the RL-based approach presented in [Mirka'20], which targets energy-efficient resource allocation in multi-core servers. The system leverages an RL agent, implemented as an NN, that selects an optimal resource allocation action, such as task arrival rates or CPU utilization, based on the current state of the system (e.g., number of cores, operating frequency). A reward based on energy efficiency metrics guides the training process, resulting in a balanced exploration and exploitation process that adapts to changing workload conditions. The results show a significant reduction in energy consumption relative to standard Linux governors.

A similar methodology is introduced in [Majumder'21] to optimize performance in Near-Memory Processing (NMP) systems. This work focuses on reducing communication overhead by remapping data and computations based on memory-related statistics, including page access rates and memory migration patterns. The RL agent, trained using a DNN, receives feedback from performance monitors embedded in the memory controller that track memory transfers. This continuous feedback loop enables adaptive tuning of memory access strategies, leading to substantial performance gains compared to baseline solutions.

Incremental learning has also been explored to predict cloud task performance. The authors in [Hilman'18] explore a hybrid incremental learning implementation of a Long Short-Term Memory (LSTM) algorithm together with a K-Nearest Neighbor (KNN) one. They state that classical batch-based models are inadequate for this scenario, while incremental approaches can adapt to evolving conditions.

Although incremental learning has been popular on high-resource computing platforms, there has been relatively little research on its use for resource-constrained environments such as edge computing systems. These platforms impose significant limitations on computational overhead and energy usage, necessitating lightweight and efficient adaptation mechanisms. Some works have addressed this gap by combining lightweight modeling techniques with run-time updating mechanisms. In [Gupta'18], the authors aim to minimize GPU power usage in mobile systems by predicting the frame processing time of graphical workloads. Their approach

involves a mathematical model that estimates the frame rendering time, followed by run-time updates of the model parameters using a Recursive Least Squares (RLS) algorithm to adapt to variations in the workload. This method dynamically adjusts the GPU frequency to meet real-time constraints while reducing energy consumption, achieving accuracy comparable to static models that require extensive prior knowledge of the workload.

A similar approach is described in [Mandal'20], where the authors implement an adaptive energy and performance management strategy for heterogeneous computing platforms. This strategy combines offline-defined resource policies with run-time model training using power and performance counters. The resulting models enable run-time adaptation of the execution policy to fit new applications, resulting in lower power consumption and reduced execution time compared to static policies.

Despite these promising results, the use of incremental learning in FPGA-based systems has mainly been limited to algorithmic implementation rather than system-level workload management. For instance, [Roorda'23] demonstrates the feasibility of implementing incremental learning algorithms directly in the FPGA fabric. Similarly, [Sousa'22] analyzed the performance benefits of executing incremental learning directly on FPGAs.

In contrast to the works seen as part of the state of the art, the proposal of this Thesis focuses on FPGA systems working in dynamically changing environments (e.g., the cloud-edge continuum). The use of incremental learning leans towards workload characterization at run time to enable self-adaptation capabilities, rather than its mere implementation in hardware.

Furthermore, the proposed work targets every layer of the continuum, from abundant-resource platforms at the cloud to resource-limited platforms at the edge. This idea falls between the two opposing visions seen in the literature. One trend targets resource-limited platforms, focusing on analytical approaches that consume fewer resources by limiting performance. Such methodology is not feasible in dynamic environments like the cloud-edge continuum, since the many factors that must be accounted for would render underperforming results. The other trend aims for high-end platforms, using complex models, such as NN, to obtain good results at the expense of resources. In the scenario of this Thesis, the use of incremental models based on lightweight ML algorithms is explored, aiming to obtain good results while wasting as few resources as possible. In addition, a resource-aware orchestrator is proposed to manage the training process, updating the models only in the case of prediction degradation, and thus minimizing the resources utilized for modeling while maintaining prediction accuracy, as opposed to current state-of-the-art methodologies that employ continuous learning strategies. A qualitative comparison between the works found in the literature and the proposal of this Thesis is presented in Table 3-2

Table 3-2: State of the Art - Incremental ML for characterization of computing platforms.

Architecture	Platform	Domain	Objective	ML Technique	Model Update
[Tian'23]	CPU	C	Power	NN	Degradation[1]
[Mirka'20]	CPU	C	Rsc.	RL + NN	Continuous
[Majumder'21]	NMP	C	Perf.	RL + NN	Continuous
[Hilman'18]	CPU	C	Perf.	LSTM + KNN	Continuous
[Gupta'18]	GPU	E	Power	RLS	Continuous
[Mandal'20]	CPU + GPU	E	Power + Perf.	P&U	Continuous
[Sousa'22]	FPGA	E	Impl.	NN	–
[Roorda'23]	FPGA	E	Impl.	RT	–
This work	FPGA	C/F/E	Power + Perf.	RTs	Degradation[2]

N/A (–) | **AND** (+) | **OR** (/)

Domain
C: Cloud computing
F: Fog computing
E: Edge computing

Objective
Impl.: The goal is to accelerate the ML algorithm itself, not to exploit it for other purposes
Perf.: Performance estimation
Power: Power estimation
Rsc.: Resource estimation

ML Technique
P&U: Design-time policy and run-time update

Model Update
Continuous: The model is continuously updated
Degradation: The model is updated when its prediction accuracy degrades

[1] Model degradation is constantly evaluated, increasing the induced overhead
[2] Model degradation is evaluated by an orchestration algorithm, minimizing the induced overhead

3.5 Offline Modeling

While the goal of this chapter is to explore incremental learning to characterize dynamic workloads, an offline, static modeling baseline has been developed first to serve as a foundation for the incremental learning strategy.

Proposed Modeling Strategy

The modeling methodology proposed in this Thesis is based on supervised learning, which essentially means that the models are fed with labeled data (i.e., the inputs of the model and its expected outputs) to learn how to predict its target variable. More specifically, the proposed methodology is based on regression models, which learn to predict the numerical value of a target variable based on their inputs. Since the purpose of this chapter is to characterize FPGA workloads, the data to model, as mentioned before, is the power consumption and execution time of the reconfigurable logic of FPGAs during the computation of hardware-accelerated workloads. Hence, the modeling process starts with a workload that has to be accelerated in the FPGA fabric. The workload management infrastructure described in Chapter 2 is in charge of that job. Its workload offloading process handles the arrival of hardware acceleration requests and their configuration and execution on the FPGA fabric, following a specific scheduling policy.[6]

During the execution of the workload, the workload registration process of the infrastructure tracks each hardware acceleration request throughout its lifetime. Moreover, the monitoring framework extracts periodic run-time power consumption and performance traces from the various combinations of kernels that are accelerated during workload execution. The entire hardware acceleration process is managed in software (i.e., user/daemon execution model) running on top of an OS. Hence, the CPU of the platform can significantly influence the overall system execution (specifically in low-end devices). Thus, in addition to task tracking and power/performance traces obtained from hardware execution, the workload registration process extracts certain CPU utilization statistics to introduce information about the CPU resource contention in the models. This CPU information includes the relative CPU utilization in terms of user-space, kernel-space, and idle time. It is elaborated from run-time CPU information extracted from the OS.

This process repeats until the entire workload is executed in hardware, generating data that can be used for modeling in the meantime. However, this data cannot be directly fed into the models since, in its raw format, it does not provide any meaningful information. A pre-processing stage is therefore conducted leveraging the task tracking mechanism, which in turn provides additional data that indicates which combinations of kernels are executed in the reconfigurable slots of the FPGA at any moment. This information enables the identification of which specific kernel combination corresponds to each generated power consumption and performance trace, as well as the particular CPU utilization of that execution window (i.e., the labels for the supervised learning process). The result is a dataset of observations, each of which contains a particular kernel combination (i.e., types of kernel functions and their respective accelerator replicas), the CPU utilization data, and its average power consumption and execution time. Please note that each run of a kernel combination

[6]For all the test campaigns presented in this chapter, the scheduling policy will be FCFS. This decision does not affect the characterization since different combinations of kernels with a variable number of replicas are executed. A more complex scheduling policy will be explored later in Chapter 4.

yields multiple observations, one for each parallel kernel instance, since every parallel executing kernel has its own execution time and average power consumption. The data contained in each observation (i.e., features and target variables) is shown in Table 3-3.

Table 3-3: Data contained in a processed observation, divided into features (i.e., inputs to the models) and target variables (i.e., outputs of the models).

Features	**Target Variables**
Kernel Under Observation (ID) [1]	PS Power Consumption (W)
# <kernel> Accelerators [2]	PL Power Consumption (W)
CPU User Space Utilization (%)	Execution Time (ms)
CPU Kernel Space Utilization (%)	
CPU Idle (%)	

[1] This feature indicates the kernel for which target variables are measured.
[2] <kernel> refers to kernel type. This feature appears once per kernel.

Once the data to be learnt is properly pre-processed, it is time to dive into the modeling part itself. To do so, it is fundamental to understand the nature of the target variables to be predicted by the models. From the discussion of Section 3.3, some insights can be extracted. On the one hand, it was observed that the power consumption variable is directly related to the number of parallel accelerators per kernel, exhibiting an additive response. In contrast, the type of kernel itself did not have much influence.

On the other hand, the execution time presented a strong dependence on the kind of kernel executed and not so much on the number of parallel accelerators per kernel. This diverse behavior suggests that the power consumption variable might require simpler models to obtain good prediction results due to its additive behavior, compared to the more complex behavior of the execution time variable. In fact, it could even be that different algorithms might fit better for the different natures of variables. To analyze it, a set of ML algorithms (specifically supervised regression algorithms) has been evaluated in this Thesis:

- **Support Vector Regression (SVR):** it involves mapping the input data to a higher-dimensional feature space, where a regression hyperplane can model the relationships in the data more effectively. This mapping is done in such a way that most data points lie within a band delimited by a margin of error [Awad'15].
- **Regression Trees (RTs):** they work by iteratively partitioning the dataset into smaller subsets to reduce their heterogeneity. This division process continues until a specified stopping condition is met. It is important to highlight that RTs are categorized as weak learners, as their performance is strongly influenced by the training data, making them susceptible to overfitting.
- **Regression Tree Ensembles (RTEs):** to address the overfitting tendency of weak learners like RTs, a common approach is to aggregate multiple instances of such

models. This method, known as an ensemble, creates a more robust learner. RTEs, specifically, combine several RTs into a collective model that improves predictive performance, improves generalization, and reduces their dependence on training data [González'20].

The modeling methodology proposed in this section is depicted in Figure 3-13.

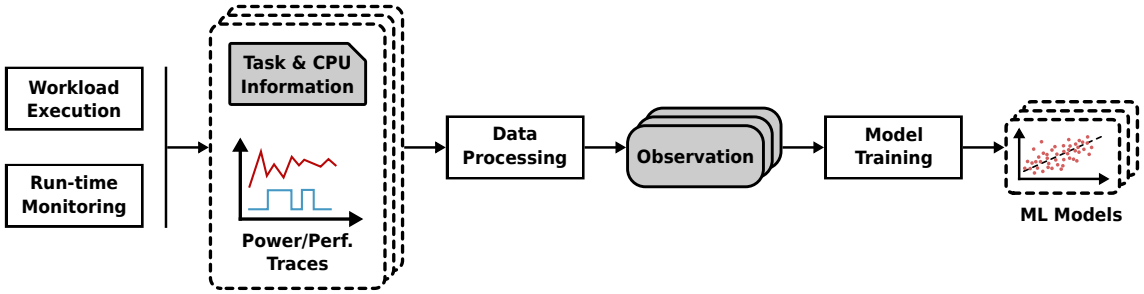


Figure 3-13: Flowchart of the modeling methodology.

Original contribution 3-2 *A device-agnostic, data-driven methodology to predict power consumption and performance in reconfigurable multi-accelerator systems.*

This offline modeling methodology will be validated in Section 3.7.1, where the proposed algorithms will be evaluated to predict the kernel interaction presented in Section 3.3. However, as it has already been discussed, this static implementation, although useful for predefined workloads, will be further extended with specific orchestration mechanisms to be effective in more dynamic scenarios, like the cloud-edge continuum scenario targeted in this Thesis. This is the incremental modeling approach described in the following section.

3.6 Incremental Modeling

The dynamic workloads present in the cloud-edge continuum require a certain degree of adaptation from the predictive models. The proposal of this Thesis tackles this need by implementing the data-driven models described in the previous section, but adapted to follow an incremental learning approach that allows them to be updated over time. However, to implement this concept effectively, a series of additional components is needed. These components are described next.

3.6.1 Extended Infrastructure

To support the use of incremental learning, a set of components has been developed and included as part of the infrastructure presented in Chapter 2. The extended infrastructure diagram is given in Figure 3-14. Note that to simplify the description of the proposed extension, most of the low-level details of the infrastructure have been removed from the diagram representation (e.g., user/daemon execution model), keeping the focus on its main building blocks. This infrastructure could be divided into two submodules: the workload manager from Chapter 2 and the adaptation module, introduced here. The main tasks of the workload manager, already described, are the hardware acceleration of the workload and its run-time monitoring. The adaptation module, on the other hand, performs three main tasks: first, it processes at run time the data coming from the workload registration process to generate meaningful observations (see the pre-processing procedure mentioned in Section 3.5); second, it creates lightweight ML-based predictive models for run-time power consumption and performance inference; and third, it maintains those models updated by re-training them at run time, following an incremental learning methodology.

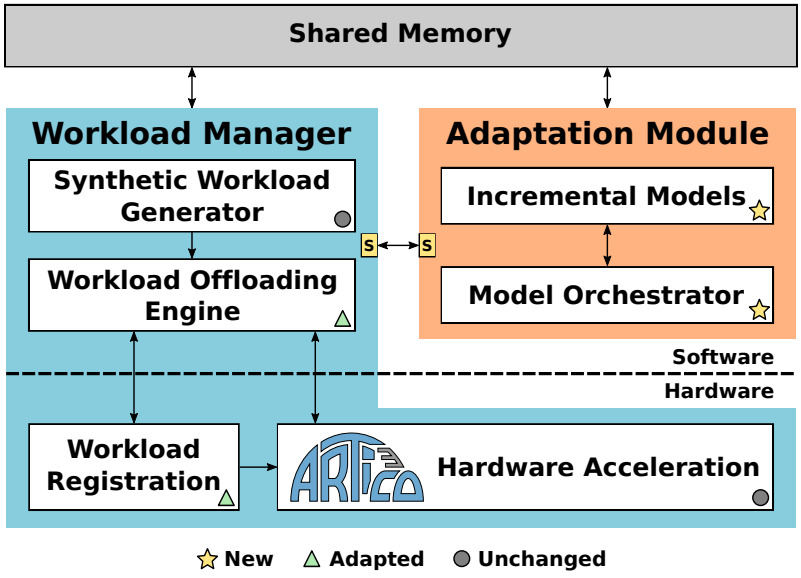


Figure 3-14: Block diagram of the infrastructure extension for incremental learning, with the unchanged, adapted, and newly created modules highlighted.

Workload Manager

The workload management infrastructure, described in detail in Chapter 2, has undergone modifications to support the effective incremental implementation of ML-based modeling. The main difference between offline and incremental learning is that training and inference are performed at run time. Therefore, the primary change that has to be applied to the workload management infrastructure is the run-time transference of monitoring data to the adaptation module for later processing and training. To do so, a set of communication channels has been implemented, connecting both subcomponents to pass information between them, including message-passing interfaces based on sockets (represented as yellow boxes in Figure 3-14) and shared memory mechanisms. Moreover, the workload offloading and workload registration processes have been adapted to leverage these interfaces for the run-time data transfers.

Adaptation Module

At run time, while the workload is deployed by the workload offloading process of the infrastructure, monitoring traces generated by the workload registration process are received by the adaptation module. The adaptation module pre-processes these monitoring data on the fly to transform them into observations such as the one described in Table 3-3. Once data is processed and observations are generated, the module feeds them to the ML-based models to train them. The model training process is coordinated by a component named the resource-aware model orchestrator. Its purpose is to maintain the accuracy of the models while reducing the training time as much as possible to keep the models lightweight, since resource availability could be heavily restricted for specific low-end devices. These functionalities are dedicated components of the adaptation module, which are described next.

Incremental Models

The monitoring data received by the adaptation module is redirected to this component, which processes it, generating the labeled observations at run time. Consequently, these observations are used to train the ML models incrementally. As the modeling algorithm, a regression tree implementation specifically tailored for incremental learning, known as the Hoeffding adaptive regression tree [Bifet'09], has been selected. The model implementation is performed as on-chip software (i.e., running on the embedded CPUs of the target platform), using an incremental learning library named River [Montiel'21]. Section 3.6.3 gives a more detailed description of this modeling strategy.

Model Orchestrator

The literature review showed that the usual implementation of incremental learning continuously updates the models as observations are processed. The primary drawback of this approach is the overhead associated with the constant generation of traces, observation processing and model training. However, it might be the case that, after a certain period of training, the models, in their current state, can accurately predict the incoming observations, meaning that additional training will not improve the prediction accuracy. Hence, a key aspect of this Thesis's proposal is to employ a resource-aware model orchestrator that can take advantage of those situations. The goal of this resource-aware model orchestrator is to maintain the model's prediction accuracy while minimizing resource utilization and the overhead induced during the process.[7] Its strategy is to decide whether to start/stop the process of incremental (re)training based on the evolution of the prediction accuracy of the model, avoiding training it blindly forever.

The model's learning process, ruled by the model orchestrator, is described next.

3.6.2 Model Orchestrator for Incremental Learning

The operation process of the model orchestrator is depicted in Algorithm 3-1. A detailed description of its main components is provided in the following sections.

Orchestration States

The model orchestrator can be in two different states: *Train*, during which the model is updated on new observations incrementally; and *Idle*, a state in which the models are kept unchanged (i.e., without training overhead). Furthermore, a *Test* stage is periodically triggered during the *Idle* state to evaluate the prediction quality of the model, determining whether to remain in *Idle* or return to *Train*.

Orchestration Operation

The orchestrator has a series of parameters to tune its operation, listed in Table 3-4. Note that those parameters modify the orchestrator's operation but have nothing to do with the models themselves; they are not hyperparameters of the models.

Besides these parameters, the following three functions guide the orchestrator:

- **Main function:** the primary model orchestrator function (line 1) initializes the state to *Train* (line 3), subsequently entering an infinite loop (line 4). When in

[7] The resource-aware qualifier of the model orchestrator refers to its effort in minimizing the resource utilization and execution overhead induced by the training process.

Algorithm 3-1 Resource-aware model orchestrator.

```
1:  function Main
2:      // Start at Train
3:      state ← Train
4:      while true do
5:          if state = Train then
6:              // Train state
7:              state ← call Train(epochs_tr, tolerance_tr, baseObs_tr, obsRedF_tr)
8:          else
9:              // Idle state -> Wait idleObs obs
10:             WaitXObs(idleObs)
11:             // Model evaluation (Test)
12:             state ← call Test(epochs_ts, tolerance_ts, baseObs_ts, obsRedF_ts)
13: function Train(epochs_tr, tolerance_tr, baseObs_tr, obsRedF_tr)
14:     for i ← 0 to epochs_tr − 1 do
15:         obsToLearn ← baseObs_tr × max(0, 1 − obsRedF_tr × i)
16:         errorDiff ← UpdateModel(obsToLearn)
17:         if errorDiff ≥ tolerance_tr then
18:             return Train
19:     return Idle
20: function Test(epochs_ts, tolerance_ts, baseObs_ts, obsRedF_ts)
21:     for i ← 0 to epochs_ts − 1 do
22:         obsToTest ← baseObs_ts × max(0, 1 − obsRedF_ts × i)
23:         errorDiff ← EvaluateModel(obsToTest)
24:         if errorDiff < tolerance_ts then
25:             return Idle
26:     return Train
```

Train, the training function is executed (line 7), and returns whether further training is required (i.e., stays in *Train*) or the prediction accuracy is good enough (i.e., changes state to *Idle*). In the *Idle* state, the algorithm waits for the time equivalent to acquire a certain number of observations ($idleObs$, line 9), skipping operations such as trace acquisition, observation processing, and model training to reduce computing overhead.[8] Afterwards, the accuracy of the model is evaluated by the *Test* function (line 12) to measure whether the accuracy of the model remains in good shape. In the event of a positive evaluation, the state remains in *Idle*. Otherwise, it changes to *Train*, restarting the incremental training phase.

[8]Since the relationship between observation and traces is complex, this time has been calculated by averaging after exhaustive experimentation.

Table 3-4: Configuration parameters used in the resource-aware model orchestrator.

Group	Parameter	Description
Common	$\texttt{baseObs}_{\texttt{tr}}$	Base number of observations used to train the model with
	$\texttt{obsRedF}_{\texttt{tr}}$	Reduction factor applied to number of observations for each training iteration within a training session
	$\texttt{baseObs}_{\texttt{ts}}$	Base number of observations used to test the model with
	$\texttt{obsRedF}_{\texttt{ts}}$	Reduction factor applied to number of observations for each test iteration within a test session
	`idleObs`	Number of observations that the training should wait in *Idle* state
Model-Specific	$\texttt{epochs}_{\texttt{tr}}$	Number of training iterations per each training session
	$\texttt{tolerance}_{\texttt{tr}}$	Acceptable prediction error variability for a training session
	$\texttt{epochs}_{\texttt{ts}}$	Number of test iterations per each test session
	$\texttt{tolerance}_{\texttt{ts}}$	Acceptable prediction error variability for a test session

- **Training function:** the training procedure (line 13) lasts until the variation of the prediction error remains stable below a certain threshold. For each of them, the model is updated incrementally, using a certain number of observations, a quantity that can change dynamically (line 15). The exact number of observations depends on a reference value ($baseObs_{tr}$) that is weighted by a factor based on the current iteration count (i) and a reduction factor ($obsRedF_{tr}$), as depicted in Equation 3-1. This approach allows the number of observations used per epoch to increase or decrease, based on that reduction factor, as the training process progresses. If the variation of the prediction error remains below a threshold ($tolerance_{tr}$, line 17) for every iteration, the training function returns *Idle* (line 19) since more training does not further improve the model. Otherwise, *Train* is returned (line 18), forcing the main function to trigger another training procedure. Therefore, the models are trained until the variation of the prediction error remains below $tolerance_{tr}$ for consecutive $epochs_{tr}$ iterations. Please note that the prediction error is computed between the model prediction and the value from the observation.

$$nextObs = baseObs_{tx} \times max(0, 1 - obsRedF_{tx} \times i) \tag{3-1}$$

- **Test function:** the test function (line 20) evaluates the accuracy of the model following an approach similar to that used in the training function. The accuracy of the model is verified; in case the prediction error goes below a threshold ($tolerance_{ts}$, line 30), this function returns *Idle* to indicate the model is performing well and no further training is needed. However, when the prediction error increases above the threshold for a consecutive number of iterations ($epochs_{ts}$), *Train* is returned to indicate further training is needed.

Model Update Synchronization

Multiple models can be managed simultaneously by the model orchestrator, which forces them to work in sync. This approach enables the models (e.g., power consumption and performance) to be trained and tested with just one set of trace acquisition, observation pre-processing and model training procedures, effectively reducing their impact on the system resources and overhead. Although the learning process of the models works in sync, different models might require treatments that are specific to them. Hence, the configuration parameters of the orchestrator have been divided into a group of standard parameters and a group of model-specific parameters, as shown in Table 3-4. The common ones are parameters like the reference number of observations used per training/testing iteration, which are shared by all models. In turn, the model-specific parameters, such as the acceptable error tolerance during training, define settings that are tailored to the specific learning process of each model.

By combining the described functions and parameters, the orchestrator retrains the models only on new observations when their prediction accuracy deteriorates. Therefore, the models are kept updated while their associated overhead and resource utilization are minimized.

Figure 3-15 shows an example representation of the model orchestrator operation with its different states and functions, during the execution of an arbitrary workload. Since collecting and processing traces one by one induces significant overhead to the system, the orchestrator performs those operations in batches (i.e., a group of traces is collected first and then used to train/test the models, as represented with the orange and green/yellow regions shown in Figure 3-15). Following this approach, the communication overhead of continuous data movement is also significantly reduced. Monitoring data is captured via the workload registration process in the acquisition phase (orange region). The monitoring data comprises a measurement of power consumption and execution time from a particular time window during the operation of the system, together with the corresponding task tracking and CPU usage information. Then, traces are turned into observations (i.e., input features and target variables to be fed to the models) using the pre-processing approach shown in Section 3.5, and fed into the models for training them (green region). Since a monitor trace stores the power consumption and performance of the system during a certain time window, various kernel combinations could be captured with a single trace. Therefore, multiple observations might be extracted from one trace.

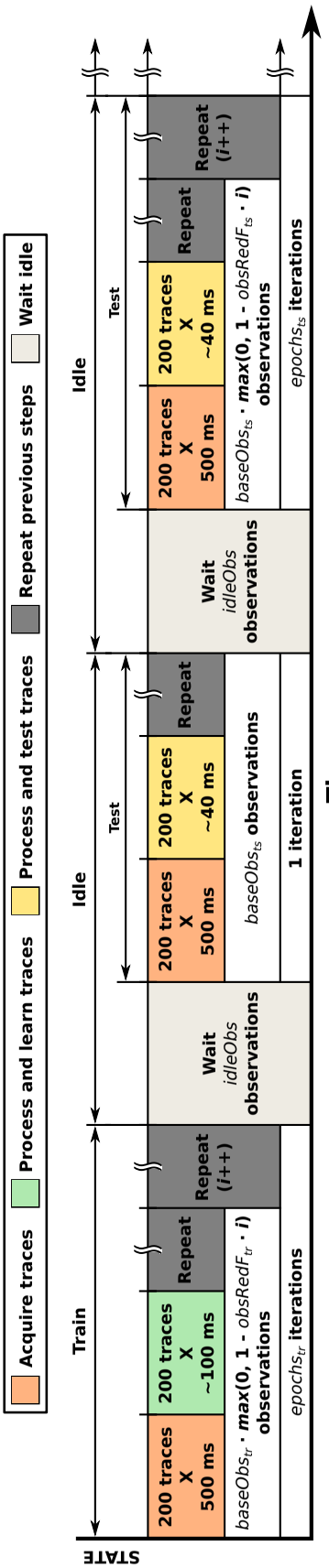


Figure 3-15: Incremental learning example with the different resource-aware model orchestrator stages.

In the representation shown in Figure 3-15, the state of the model orchestrator starts in *Train*. In this *Train* state, monitoring data is collected every 500 ms (first orange region), ultimately collecting 200 traces (this particular spacing between captures and the number of them being captured are selected since they produced the best results after exhaustive experimentation). Monitoring data is then transferred to the adaptation module, where it is processed into observations that are used for incremental training (first green region). This sequence continues for $baseObs_{tr}$ trained observations, which is considered the first *Train* iteration. This training process is repeated during $epochs_{tr}$ iterations, varying, after each iteration, the number of observations to be trained (see Equation 3-1). When all iterations are completed, and if the training error is below the $tolerance_{tr}$ threshold in each of them (the example in the figure assumes this to be true for the first *Train* period), the model orchestrator state goes to *Idle*. If, instead, after any iteration the error exceeds $tolerance_{tr}$, the state would remain in *Train*, and the *Train* process would restart.

In contrast, in the *Idle* state, the adaptation module remains on standby without consuming resources or inducing overhead until the time it would take to capture $idleObs$ observations has elapsed (first beige region).[9] Then, the accuracy of the model is evaluated in the *Test* function. Two hundred monitoring traces are collected (second orange region), processed into observations, and the model is tested by comparing its prediction with the actual measured value (first yellow region). This test process is repeated for $baseObs_{ts}$ observations, which is considered a test iteration. The prediction error is evaluated against $tolerance_{ts}$ after each iteration. If it is below that threshold, the state of the model orchestrator returns to *Idle* since the model predictions are still accurate enough. This scenario is represented in the first *Test* phase of the example. In turn, when the prediction error rises above $tolerance_{ts}$, the testing process is repeated to check whether the prediction deviation is a transient event. The process would repeat for a maximum of $epochs_{ts}$, updating the number of observations to be tested after each iteration (Equation 3-1). This case is illustrated in the second *Test* phase of the example. If the prediction error remains over $tolerance_{ts}$ after the maximum number of iterations, the model orchestrator requests a new training phase by setting its state again to *Train*. Otherwise, the state will remain in *Idle*. This entire process is repeated indefinitely (i.e., until complete workload execution), keeping the model updated over time while minimizing resource utilization and induced overhead when compared to alternative continuous training solutions.

Since the specific selection of the configuration parameters of the model orchestrator can significantly impact the success of this approach, Section 3.7.2 presents an exhaustive sensitivity analysis of the parameters.

Original contribution 3-3 *A resource-aware model orchestration mechanism to perform run-time adaptive incremental learning and keep models updated while minimizing system overhead.*

[9] Since the relationship between observation and traces is complex, this time has been calculated by averaging after exhaustive experimentation.

3.6.3 Incremental Modeling Strategy

The modeling strategy for this incremental learning implementation of the models is similar to the one described in Section 3.5. During the workload execution managed by the workload management infrastructure, the workload registration process collects run-time monitoring data (i.e., power/performance traces, tasks tracking data, relative CPU utilization). That monitoring data is used to build the observations described in Table 3-3. However, the primary difference is that the raw monitoring data captured at run time is transferred on the fly to the adaptation module, in which it is effectively pre-processed into the mentioned observations. Those observations are then fed to the incremental ML models, in a training process that the resource-aware model orchestrator rules. This entire process continues until the workload has been completely executed, which might be forever in a real cloud-edge continuum scenario. This modeling strategy, entirely ruled by the resource-aware model orchestrator described in the previous section, is depicted in Figure 3-16.

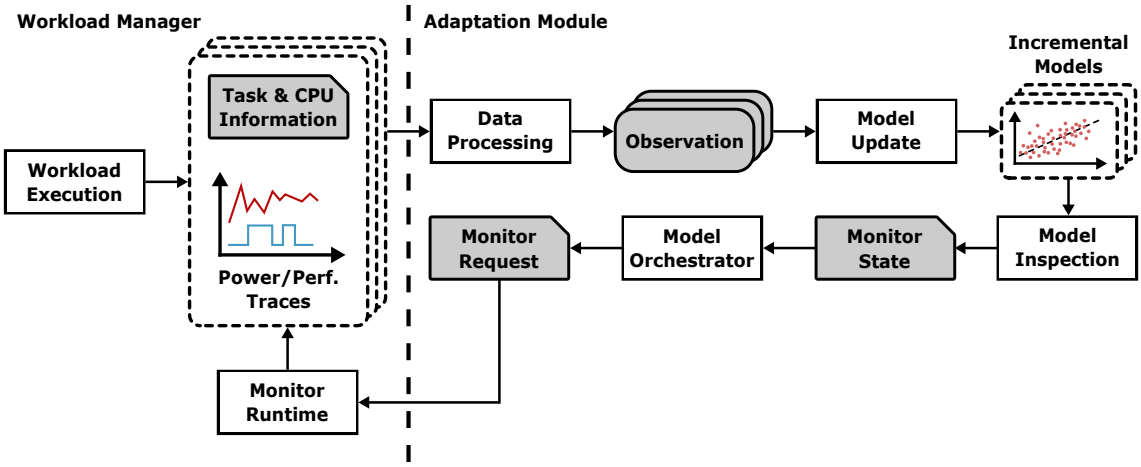


Figure 3-16: Flowchart of the incremental modeling methodology.

The actual implementation of the ML-based models in this Thesis is based on a library for incremental learning implementation named River [Montiel'21]. Its key advantage lies in its lightweight models, which include built-in adaptation mechanisms to detect and overcome model drifting. Regarding the ML algorithms used for modeling, regression trees have been the selected candidate, since they can learn from massive data streams, which is a key in lifelong incremental learning scenarios, and are particularly well suited for dynamic environments like the one faced in this Thesis [Hulten'01].[10] In fact, the selected candidate is the Hoeffding adaptive regression tree [Bifet'09], a regression tree variant specifically tailored for incremental learning. The model drift detectors embedded in this algorithm track the prediction accuracy of each tree branch, replacing them with new ones when their accuracy decreases, which makes it ideal for the role.

[10]The use of regression trees is also backed by the promising results obtained with them in the offline modeling scenario, which will be analyzed in (Section 3.7.1 later in this chapter).

Since the additive nature of the power consumption variable makes it more predictable (as will be discussed in Section 3.7.1), a simple Hoeffding adaptive regression tree is being used for the implementation of power consumption models. Conversely, the performance variable, which depends strongly on the kernel type to be executed, presents a much higher prediction complexity; therefore, it has been modeled using an ensemble of these Hoeffding adaptive regression trees [Gomes'18], as groups of learners build up stronger learners compared to individual ones.

Please note that a classic ML algorithm like the regression tree has been selected over more complex ones (e.g., NNs) to minimize the induced overhead, which is a requirement on edge platforms. Moreover, as will be demonstrated in Section 3.7.1, classical approaches have provided enough prediction accuracy in these scenarios, making more computationally expensive solutions detrimental.

Original contribution 3-4 *An incremental data-driven modeling extension to predict power consumption and computing performance in reconfigurable multi-accelerator systems at run time and on demand.*

3.7 Validation

The proposed data-driven modeling methodology has been validated in three stages. First, the modeling methodology itself has been validated as an offline implementation, evaluating the effectiveness of using an ML-based approach for predicting power consumption and execution performance in FPGA systems. Then, the incremental learning extension of this modeling methodology has been evaluated under a dynamic workload. The section ends with an assessment of the degree of generalizability and portability achieved with the proposed modeling methodology. In all cases, the MachSuite benchmarks described in Section 2.5.1 have been used as the applications to be modeled.

3.7.1 Offline Modeling

The evaluation of the offline implementation of the proposed modeling methodology starts by describing the experimental setup used to carry out the validation campaign. Then, a brief section introduces a comprehensive comparison between different ML algorithms used to implement the models, selecting the one that provides the best results. Finally, the results obtained with the best-performing models are evaluated.

Experimental Setup

The setup for this experimental campaign, as every validation section in this Thesis, uses the workload management infrastructure presented in Chapter 2, which has been implemented on a Zynq UltraScale+ ZCU102 evaluation board.[11] This specific platform is a heterogeneous system that combines hardware (i.e., hardware accelerators running at 100 MHz in the PL) and software (i.e., high-performance and real-time CPUs running at 1.2 GHz in the PS) execution on-chip, without the need for soft-core alternatives that render lower performance and consume reconfigurable resources that could be used for other means. Furthermore, the broad peripheral and interface support of this board, together with high-speed connectivity ports, makes it suitable for a wide range of applications and domains, which has led to its widespread use in both industry and academia. It is therefore a good fit for this work.

The number of reconfigurable slots that fit in any given platform is limited by the logic resource constraints of that reconfigurable device. Hence, for this particular setup, the maximum number of parallel hardware accelerators is restricted to eight.

A set of synthetic workloads has been created with the synthetic workload generator for this evaluation process. The configuration of each workload, governed by the adjustment of the five configuration parameters available in the generator, is depicted in Table 3-5. The N, K_P, and A_R configuration parameters are kept the same, ensuring that each workload contains the same number of hardware acceleration requests, using the same pool of kernels and an equivalent data-level parallelism (by having the same number of parallel accelerators). In turn, the workloads vary in the rate of arrival of new hardware acceleration requests (mean inter-arrival time $\overline{T_I}$) and the computational load (job size J_R). Please note that every kernel included in this workload is extracted from the MachSuite benchmarks described in Section 2.5.1.

Table 3-5: Workload Variations.

	N	K_P	A_R	$\overline{T_I}$	J_R
W_0	20,000	11 MachSuite Kernels	[1,8]	10 ms	[8,128]
W_1				10 ms	[128,2048]
W_2				100 ms	[8,128]
W_3				100 ms	[128,2048]
W_4				500 ms	[8,128]
W_5				500 ms	[128,2048]

During the execution of each workload variation (i.e., each entry in Table 3-5), the monitoring framework collects traces that result in between 5,000 and 25,000 meaningful observations after pre-processing (i.e., labeled observations containing power consumption and performance measurements of kernel combinations). These observations are used to create a dataset that is employed in the remainder of this

[11] XCZU9EG-FFVB1156-2-I device.

section to perform an offline (i.e., not on-chip at run time) validation of the proposed modeling methodology. Note that the target evaluation board exposes differentiated power rails for the PS and PL. Hence, the obtained monitoring traces are from PS power consumption, PL power consumption, and execution time. Regarding the fluctuation in the number of meaningful observations generated, this variation depends on the configuration parameters that differ between workloads, i.e., the mean inter-arrival time and the job size.

For the mean inter-arrival time, the higher it is, the longer it takes for the workload to be executed, therefore leading to a higher number of measurements (remember, they are collected periodically at a fixed rate). However, the execution of kernels is also more spaced, resulting in periods with no kernel execution and traces that are not meaningful. Conversely, when the mean inter-arrival time decreases, while the number of measurements is reduced, it is more likely to measure actual kernel executions that are meaningful. In turn, the variation in the job size directly affects the time that a kernel is in execution. Therefore, an increase in job size makes kernels execute for longer periods, increasing the probability of i) the monitoring framework capturing the execution of the kernel, obtaining more meaningful observations; and ii) multiple kernels being executed simultaneously, collecting observations that represent more complex kernel interactions.

Modeling Results

This section briefly evaluates how the proposed ML models perform when predicting power consumption and execution performance for each of the generated workload variations. As previously mentioned, the dataset generated per workload variation contains observations with PS power consumption, PL power consumption and execution time traces. To employ these observations for modeling, each of them has been conveniently labeled with an identifier representing the kernel under evaluation, together with the specific kernels that are executed simultaneously and the number of replicas of each of them at the instant of capture of the observation (see Table 3-3 for details on observation fields). The set of models presented in Section 3.5 has been trained offline with the generated datasets for each of the workload variations.[12]

To evaluate the prediction accuracy of the proposed models, a k-fold cross-validation approach has been followed, specifically with five folds. This strategy, commonly used in the ML field, consists of dividing the dataset into k subsets (five in this case). It is an iterative process in which, for each iteration, a different subset is used to evaluate the prediction accuracy of the model (i.e., test phase). In contrast, the remaining ones are used for training, repeating the process until every subset has been used for testing. The prediction accuracy of the models has been measured using the Root Mean Squared Error (RMSE) and Mean Absolute Percentage Error (MAPE) error metrics, calculated as shown in Equation 3-2 and Equation 3-3, respectively.

[12]The hyperparameters of each evaluated model have been fine-tuned following a grid search approach to optimize prediction accuracy.

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2} \tag{3-2}$$

$$MAPE = \frac{100}{N}\sum_{i=1}^{N}\frac{|y_i - \hat{y}_i|}{y_i} \tag{3-3}$$

In these equations, N represents the number of observations, y is the actual measurement, and $\hat{y}$ is the prediction of the model.

Table 3-6 reports the RMSE and MAPE error metrics obtained with the three ML models when predicting the PS power consumption, PL power consumption, and execution time for the six workload variations presented in Table 3-5. The results show that the proposed ML algorithms present relatively low error metrics when predicting each target variable for all workload variations, which supports the initial hypothesis that data-driven modeling methodologies might be a promising solution for predicting power consumption and performance in FPGAs. For the sake of clarity during the analysis of the obtained results, a single workload variation will be selected to focus the discussion on it in the remainder of this chapter. Similarly, just one ML algorithm will be used to model each target variable, the one reporting the best results.

Precisely, W_1 has been the workload variation selected for the validation campaigns. The rationale behind this is that the workload variation W_1 features the lower mean inter-arrival time together with the largest job size. This combination of parameters, as described in Section 3.7.1, results in the workload with the most meaningful yet complex observations, allowing the evaluation of the models in the harshest kernel interaction conditions.

Regarding the selection of the ML algorithm, focusing on their response under W_1 in Table 3-6, it can be observed that every model shows a considerably good prediction accuracy when modeling both power consumptions, rendering similar low values for both error metrics. In contrast, the RTE algorithm has slightly better accuracy when predicting execution time than the other algorithms. This behavior seems reasonable, since, as discussed in Section 3.5, the power consumption variable has a simpler additive nature that every proposed algorithm can effectively model. In turn, the execution time variable is significantly affected by the kernel type executed rather than the number of parallel accelerators, which resembles a decision tree structure, explaining the better accuracy of the RTE alternative.

Consequently, to predict power consumption in both the PS and PL, even though every proposed algorithm would yield similar results, it has been decided to use the SVR model since, for a similar RMSE, the MAPE metric is slightly better. In contrast, the prediction of the execution time is handled by the RTE algorithm. The error metrics obtained with these selected models when modeling the chosen W_1 workload variation are summarized in Table 3-7.

Table 3-6: Prediction errors for the three ML algorithms when modeling the six workload variations proposed (smaller is better).

Workload	Variable	Model	RMSE	MAPE
W_0	PS Power Consumption	RT	0.0439	1.746
		RTE	0.0408	1.641
		SVR	0.0423	1.475
	PL Power Consumption	RT	0.0076	1.684
		RTE	0.0067	1.553
		SVR	0.0069	1.519
	Execution Time	RT	0.4681	8.935
		RTE	0.3389	6.239
		SVR	0.3164	8.445
W_1	PS Power Consumption	RT	0.0473	1.593
		RTE	0.0435	1.542
		SVR	0.0447	1.329
	PL Power Consumption	RT	0.0073	1.562
		RTE	0.0070	1.526
		SVR	0.0071	1.486
	Execution Time	RT	0.4113	6.904
		RTE	0.3549	6.335
		SVR	0.3590	7.307
W_2	PS Power Consumption	RT	0.0479	1.989
		RTE	0.0449	1.843
		SVR	0.0470	1.719
	PL Power Consumption	RT	0.0066	1.614
		RTE	0.0070	1.633
		SVR	0,0070	1.623
	Execution Time	RT	0.3061	4.671
		RTE	0.2553	4.363
		SVR	0.2504	6.999
W_3	PS Power Consumption	RT	0.0459	1.614
		RTE	0.0464	1.654
		SVR	0.0494	1.500
	PL Power Consumption	RT	0.0070	1.613
		RTE	0.0067	1.554
		SVR	0.0069	1.518
	Execution Time	RT	0.5167	11.367
		RTE	0.4783	11.200
		SVR	0.5057	11.760
W_4	PS Power Consumption	RT	0.0282	1.016
		RTE	0.0229	0.913
		SVR	0.0223	0.685
	PL Power Consumption	RT	0.0072	1.855
		RTE	0.0067	1.773
		SVR	0.0067	1.712
	Execution Time	RT	0.1330	2.980
		RTE	0.1088	2.809
		SVR	0.1038	5.266
W_5	PS Power Consumption	RT	0.0468	1.837
		RTE	0.0466	1.844
		SVR	0.0514	1.818
	PL Power Consumption	RT	0.0093	1.968
		RTE	0.0227	1.999
		SVR	0.0227	2.010
	Execution Time	RT	0.5419	15.256
		RTE	0.4192	14.871
		SVR	0.4430	13.296

Table 3-7: Prediction errors for workload variation W_1 (smaller is better).

Model	RMSE	MAPE
PS Power Consumption (SVR)	0.0447	1.329
PL Power Consumption (SVR)	0.0071	1.486
Execution Time (RTE)	0.3549	6.335

Evaluation of the Results

In addition to the error metrics presented in the previous section, the graphic representation of the prediction accuracy of the models can be seen in Figure 3-17, Figure 3-18, and Figure 3-19 for the PS power consumption, PL power consumption and execution time, respectively. Each figure is composed of two graphs: the top graphs show the predicted value of each observation against the actual measured value, so that the dotted diagonal line represents the observations with perfect predictions; a close-up view of the top graphs is represented in the bottom graphs.

Please bear in mind that even though it seems that a considerable number of observations are significantly far from the dotted line, the graphs represent about 5,000 test observations each. In reality, the actual number of observations far away from the dotted line is a minority compared to the total number of observations, but it feels the other way around because most observations superpose within a tiny region close to the dotted line (as indicated by the low error metrics of Table 3-7).

Observing the graphical representation of the models, a pattern of horizontal stripes can be identified, which is caused by two factors. First, since these experiments are running on an OS, its run-time variability causes multiple equivalent observations (i.e., same combinations of kernels with an identical number of replicas) to render slightly different execution times or power consumptions. Given that a model predicts the same value (i.e., output) for the same features (i.e., inputs), multiple equivalent observations would result in the same exact prediction even though the actual measurements are slightly different, which generates those horizontal strips. On the other hand, there is a phenomenon that primarily affects power consumption (that is why those stripes are more apparent in these models), which is the fact that during workload execution, the temperature of the CPUs and the FPGA fabric changes over time. This phenomenon causes equivalent observations to render slightly different power consumptions over time, contributing also to the mentioned horizontal stripes.[13] Please note that the vertical stripes seen in both power consumption models occur because of the discrete nature of the power consumption measurements. Since the ADC used to measure power consumption has a specific resolution, the actual measurements fall in multiples of that resolution value, generating the vertical lines. However, as the error metrics in Table 3-7 demonstrate, this phenomenon does not affect the accuracy of the models.

[13] Up to 10% power consumption variations have been measured in the PL over time.

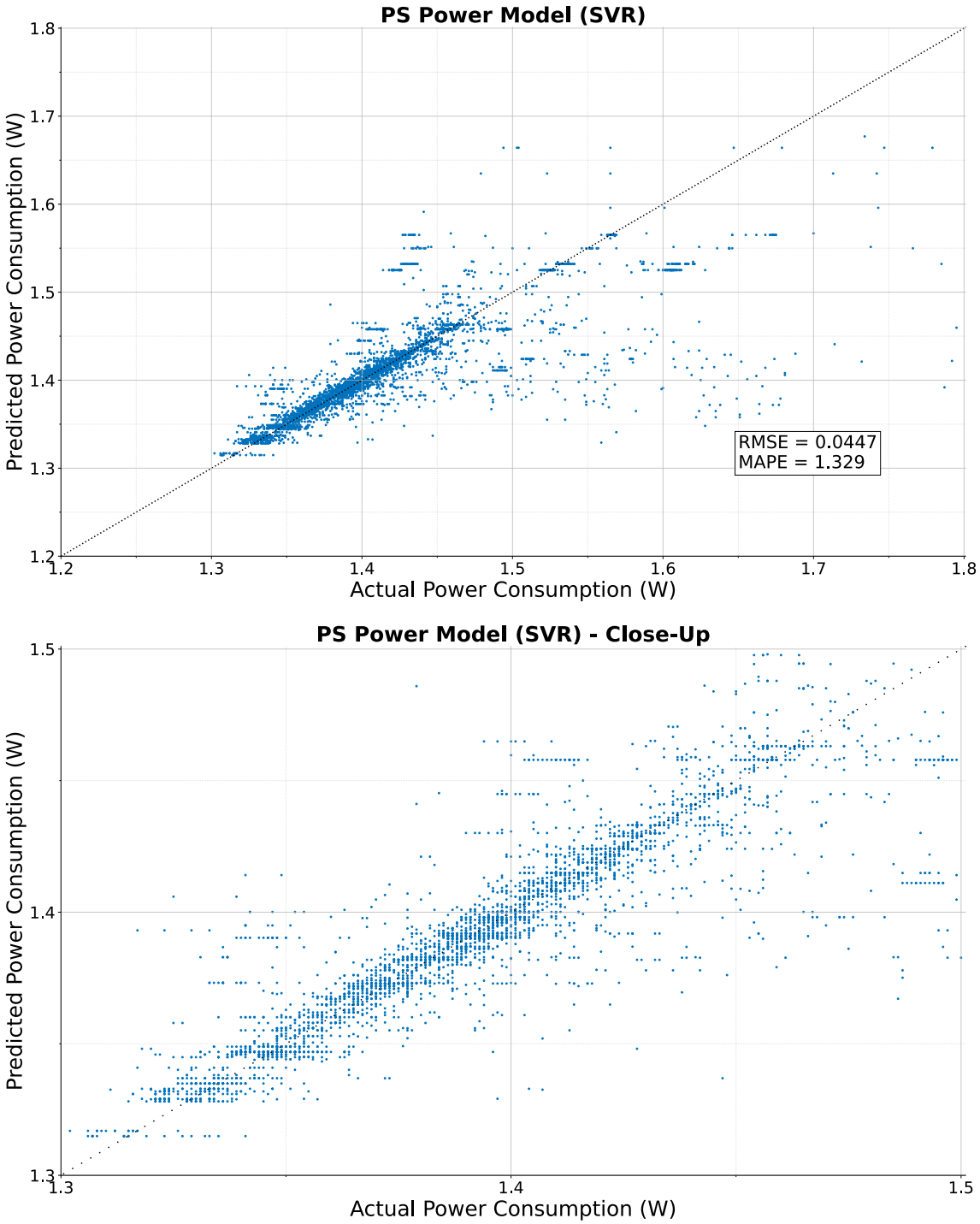


Figure 3-17: PS power consumption model evaluation.

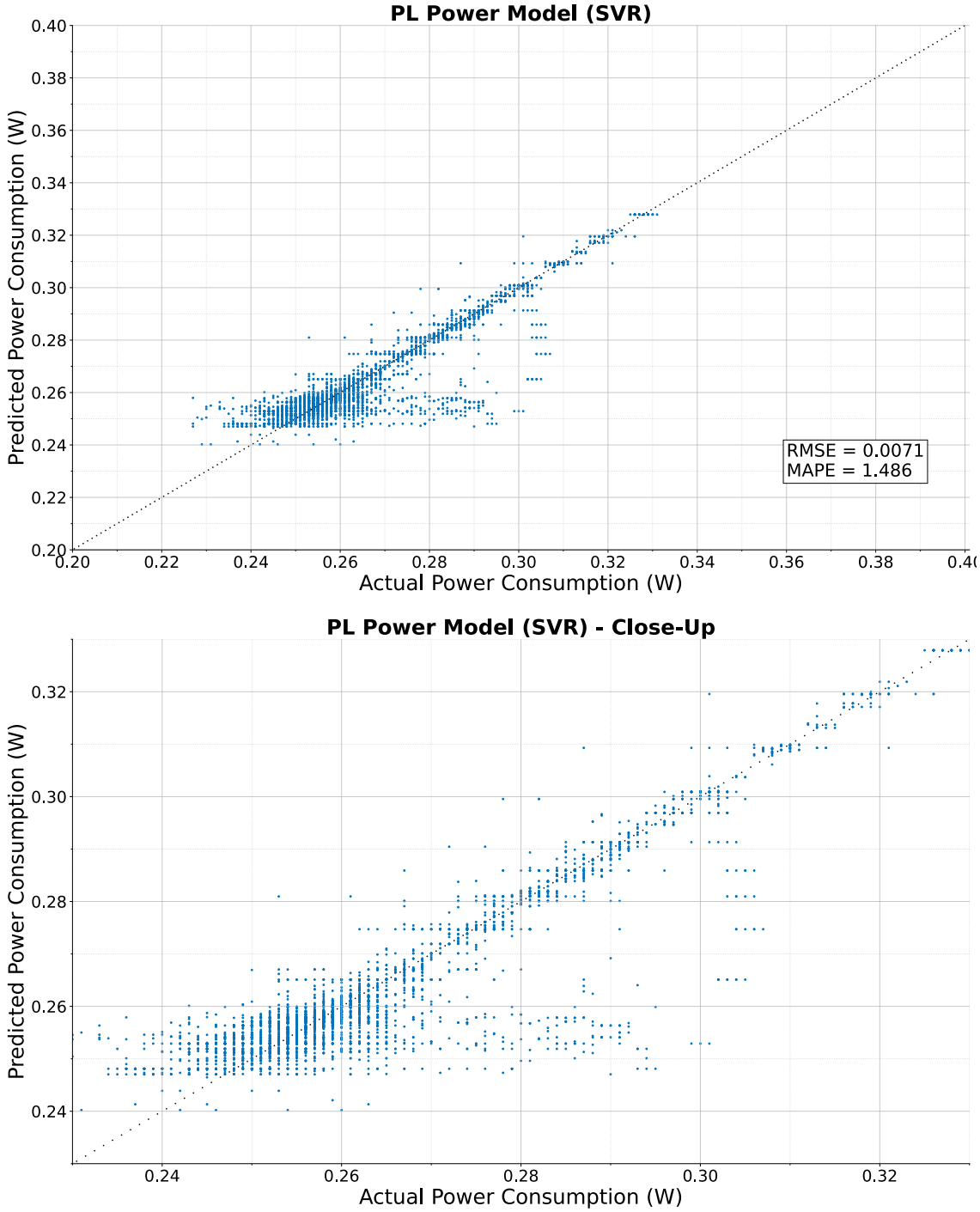


Figure 3-18: PL power consumption model evaluation.

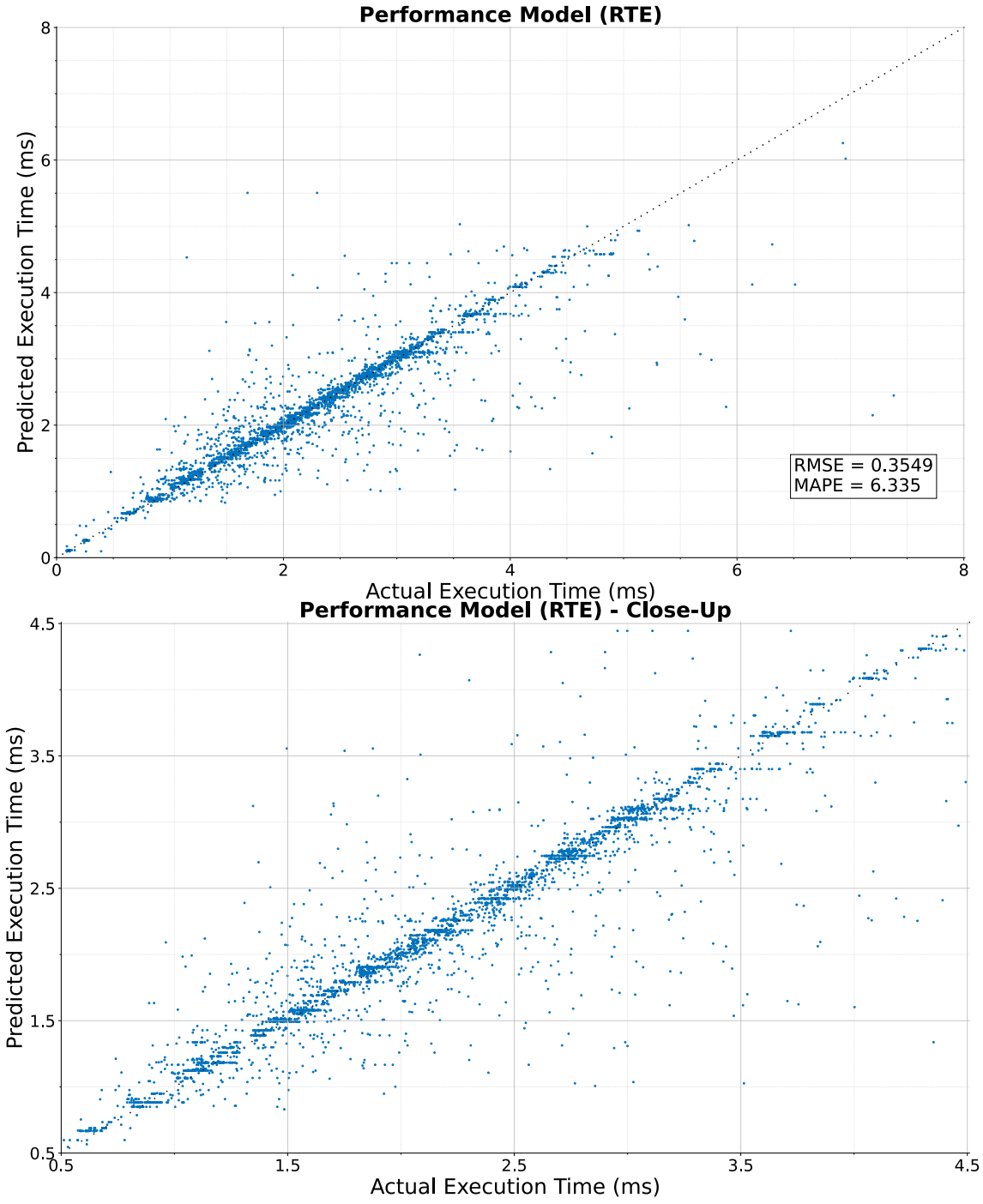


Figure 3-19: Performance model evaluation.

Next, a specific analysis of the relative error between predicted and real data is provided for both the single- and multi-kernel scenarios.

Single-Kernel Scenario

In a single-kernel scenario, the relative error between the predicted value and the real value is presented in Figure 3-20, Figure 3-21 and Figure 3-22 for the PS power consumption, the PL power consumption and the execution time, respectively. The results shown in these figures are presented by kernel type and number of accelerators.

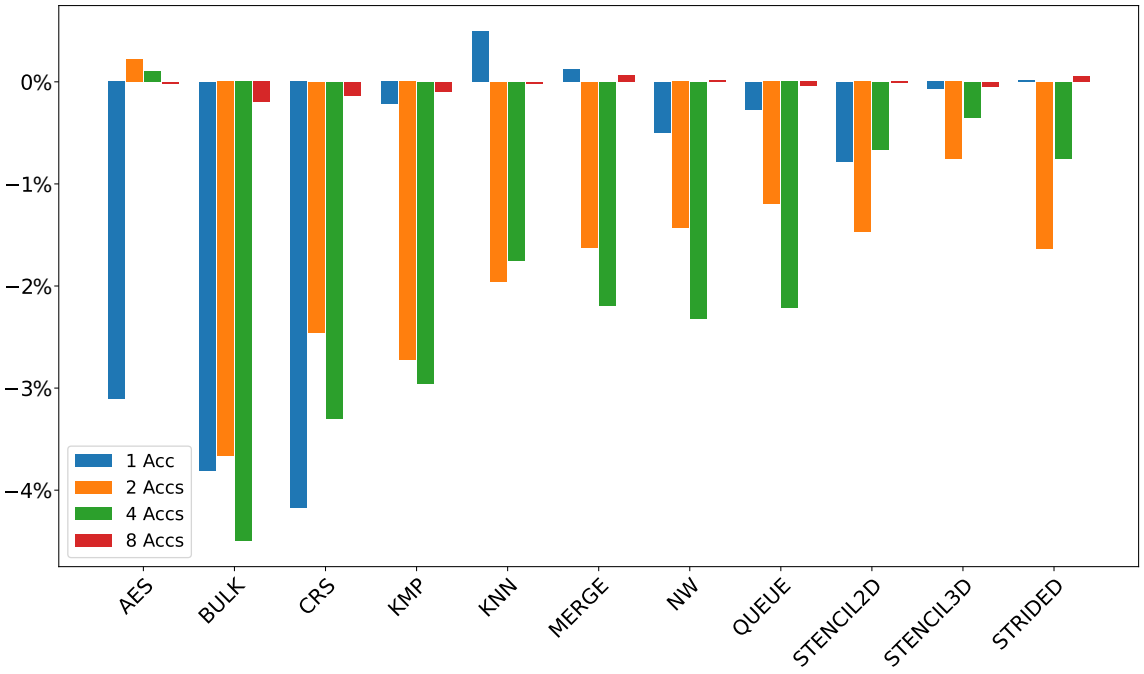


Figure 3-20: Relative error in average PS power consumption prediction according to kernel type and number of accelerators.

The results show that the prediction accuracy of the proposed models is significantly good, obtaining relative errors of less than 5% when predicting PS power consumption (Figure 3-20), less than 2% when predicting PL power consumption (Figure 3-21) and less than 5% when predicting execution time (Figure 3-22). The exception is the AES kernel with just one accelerator that the model predicts with a 9% relative error. The reason is that AES is the kernel with the fastest execution time, which makes it less probable to be captured by the monitoring framework and therefore has a more complex behavior to learn by the models. The same happens to other fast kernels in Figure 3-22 (e.g., AES, BULK, CRS, and QUEUE), which show a relative error in execution time prediction higher than the rest.

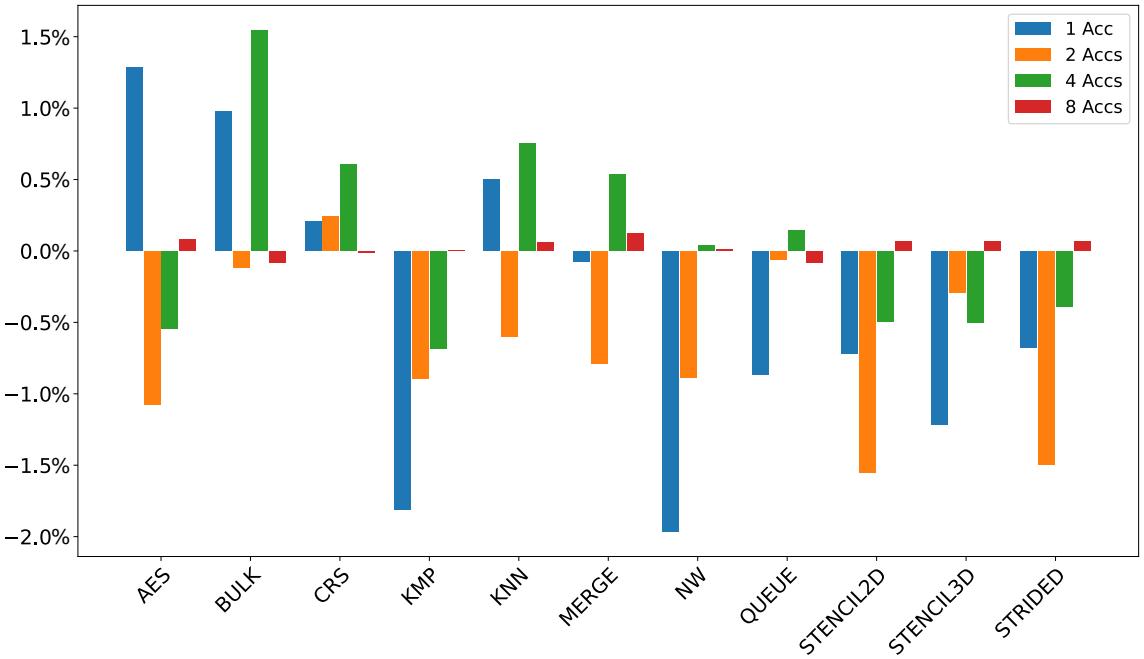


Figure 3-21: Relative error in average PL power consumption prediction according to kernel type and number of accelerators.

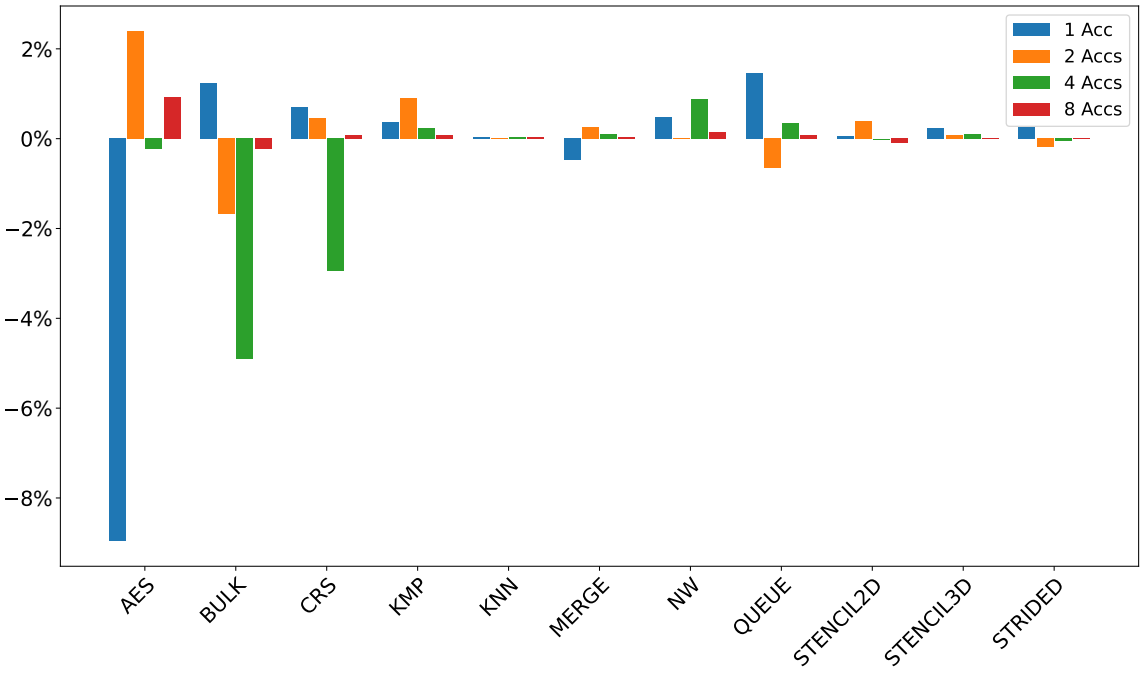


Figure 3-22: Relative error in execution time prediction according to kernel type and number of accelerators.

Multi-Kernel Scenario

Let us now assess the ability of the proposed models to predict the kernel interaction described in Section 3.3.2. For the sake of clarity in the graphical representation and associated discussion, this section will also be based on a two-kernel scenario; nevertheless, keep in mind that the discussion extends to more complex multi-kernel configurations (these scenarios are already included in the results shown in Table 3-7).

Figure 3-23, Figure 3-24 and Figure 3-25 are analogous to the ones presented in the kernel interaction analysis in Section 3.3.2 (i.e., Figure 3-10, Figure 3-8, Figure 3-9). However, in this case, these figures represent in each cell the relative error between the value predicted by the models and the real data, respectively, for the PS power consumption, the PL power consumption and the execution time. Please note that cells marked as "*" represent a particular combination of kernels that the model can predict, but there is no actual measured value to compare with.

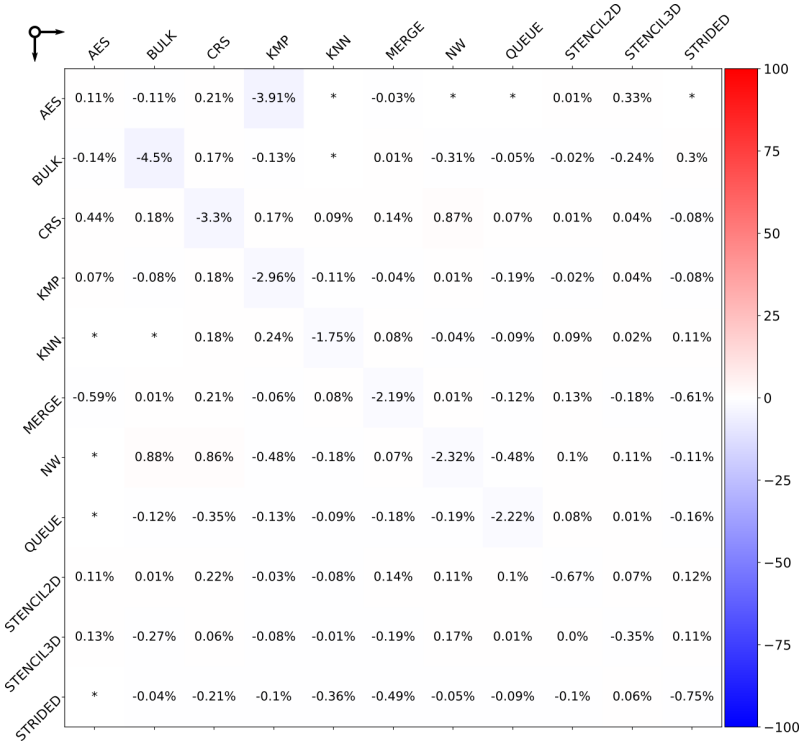


Figure 3-23: Relative error in predicting the kernel interaction impact on PS power consumption (y-axis: kernel under evaluation, x-axis: kernel executed in parallel to it).

The results represented in these figures demonstrate that the proposed models are able to predict the impact of kernel interaction in the PS power consumption (Figure 3-23) and PL power consumption (Figure 3-24) with almost 0% error relative to the actual measurement values in each combination of kernels. Even more relevant is the outcome of the model that predicts the impact of kernel interaction in the execution time (Figure 3-25), showing a prediction error relative to the actual data of less than 5% (close to 0% in most kernel combinations), except for three specific

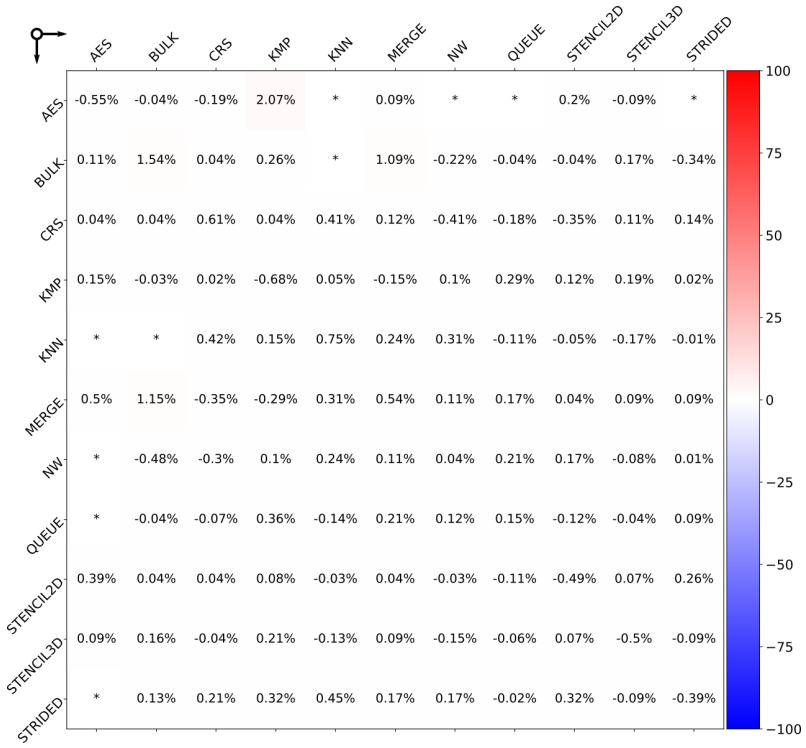


Figure 3-24: Relative error in predicting the kernel interaction impact on PL power consumption (y-axis: kernel under evaluation, x-axis: kernel executed in parallel to it).

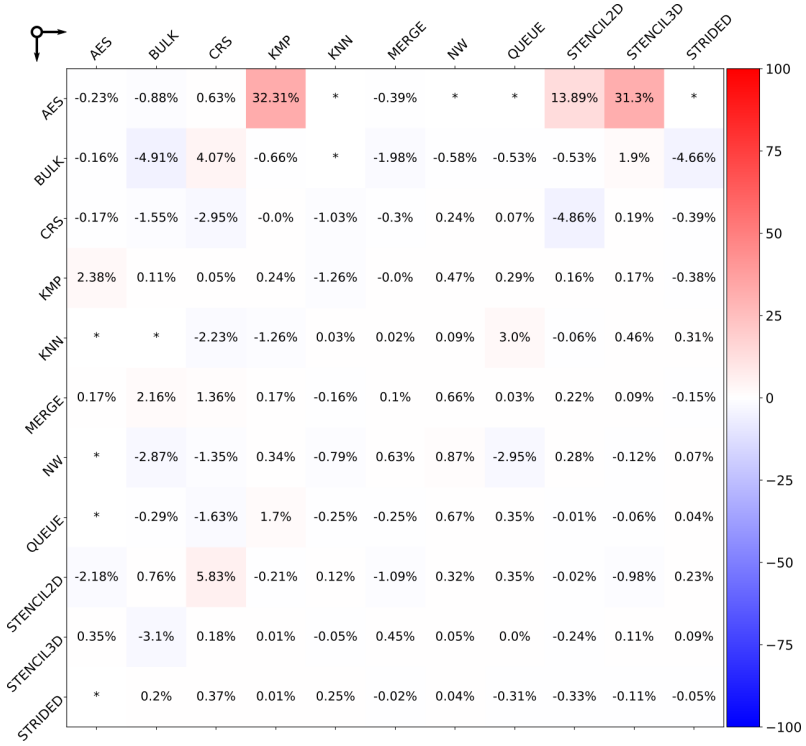


Figure 3-25: Relative error in predicting the kernel interaction impact on execution time (y-axis: kernel under evaluation, x-axis: kernel executed in parallel to it).

kernel combinations featuring the AES kernel. The reason here is that, as discussed in the single-kernel scenario, since AES executes relatively fast, there are only a few captured traces including it. This limited number of observations containing the AES kernel affects the effectiveness of the model when predicting the kernel interaction in combinations of kernels that contain AES.

A summary of these results is provided in Table 3-8. It basically collects all the kernel combinations shown in the interaction maps presented in this section (i.e., Figure 3-23, Figure 3-24, Figure 3-25) and calculates the mean and standard deviations. Hence, the table presents the mean and standard deviation of the relative prediction error of each model when modeling the multi-kernel scenario, particularized for only 2 parallel kernels. Please note that the metrics for the performance model have also been presented without including the AES kernel to evaluate the model, leaving aside the three combinations of kernels that present significantly worse accuracy.

Table 3-8: Relative error in the prediction of the impact of kernel interaction (2 parallel kernels).

Model	Mean	Std. Deviation
PS Power Consumption (SVR)	-0.22%	0.82%
PL Power Consumption (SVR)	0.09%	0.36%
Execution Time w/o AES (RTE)	-0.16%	1.44%
Execution Time (RTE)	0.54%	4.65%

The results show a mean relative error of less than 0.25% (expressed in absolute terms) when predicting power consumption in both the PS and the PL, with a standard deviation of less than 1%. In turn, a mean relative error of -0.16% and a standard deviation of 1.44% are obtained when predicting the execution time, leaving aside the AES kernel. Including it, the mean relative error rises to 0.54% with a standard deviation of 4.65%, which is still a fairly reasonable set of values.

3.7.2 Incremental Learning

This section covers the evaluation of the incremental extension of the proposed modeling methodology. The experimental setup used for this validation campaign is described first. Then, a sensitivity analysis of the configuration parameters of the model orchestrator is provided. The section ends with a thorough evaluation of the results obtained with this incremental learning methodology.

Experimental Setup

The experimental setup used for this set of tests is essentially the same as the one employed in Section 3.7.1: the Zynq UltraScale+ ZCU102,[14] the MachSuite benchmarks, and the same target variables (i.e., PS power consumption, PL power consumption, execution time). The only difference is in the process of generating the workload used for validation.

Since the purpose of this validation campaign is to evaluate the accuracy of incremental learning in a cloud-edge continuum scenario, the workload to perform the validation must have the nature of a dynamic workload (see Section 1.4.2). To do so, the validation workload, generated with the synthetic workload generator, contains 60,000 hardware acceleration requests, incorporating additional kernel functions gradually over time:[15] the first 20,000 hardware acceleration requests feature only 4 kernels, the following 20,000 include another 4 additional kernels for a total of 8, and the last 20,000 include all the 11 available kernels. The purpose behind this synthetic workload is to prove the ability of the proposed modeling methodology to adapt to new tasks while maintaining previously learnt knowledge. The scheduling policy used throughout these experiments is an FCFS.

Sensitivity Analysis of the Model Orchestration Parameters

The configuration parameters of the model orchestrator (see Table 3-4) have a significant impact on the prediction accuracy of the models. Hence, they must be fine-tuned to obtain optimal models. Using a trial-and-error approach to evaluate every possible combination of parameters would be very time-consuming and render suboptimal solutions. Hence, an exhaustive sensitivity analysis has been conducted to choose the best trade-off configuration parameters.

The sensitivity analysis follows a grid search approach in which the modeling strategy presented in Section 3.6.3 has been used to model multiple times the synthetic workload described in the previous section, changing in each iteration the selection of orchestrator parameters. The goal of this sensitivity analysis is to find the combination of parameters that provides the highest prediction accuracy but uses the fewest number of observations for training, as reducing the training time reduces the use of computing resources and the overhead during the trace acquisition, observation processing, and model training stages that need to be performed at run time. Based on preliminary tests, the search space for the orchestrator parameters has been limited to the options listed in Table 3-9. Each grid search candidate (i.e., specific selection of orchestrator parameters) has been evaluated on two metrics: a MAPE prediction error metric (see Equation 3-3) and the percentage of observations used for training over the total observations.

[14] XCZU9EG-FFVB1156-2-I device.

[15] The synthetic workload generator also sets the number of parallel replicas per kernel by sampling from a uniform distribution of powers of two (up to 8 replicas, the maximum number of reconfigurable slots available in the experimental setup).

Table 3-9: Search space for the model orchestrator parameters.

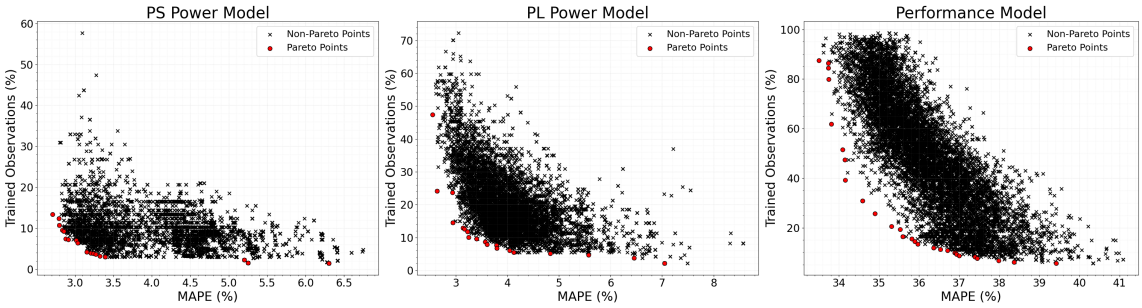
Nature	Parameter	Evaluated Values
Common	$\texttt{baseObs}_{\texttt{tr}}$	[500, 1000, 2000]
	$\texttt{obsRedF}_{\texttt{tr}}$	[0, 0.2, 0.4]
	$\texttt{baseObs}_{\texttt{ts}}$	[100, 200, 300, 400]
	$\texttt{obsRedF}_{\texttt{ts}}$	[0, 0.2, 0.4]
	idleObs	[500, 1000, 2000]
Model-Specific	$\texttt{epochs}_{\texttt{tr}}$	[2, 3, 4, 5]
	$\texttt{tolerance}_{\texttt{tr}}$	[1, 2, 3, 4, 5, 6, 7, 8]
	$\texttt{epochs}_{\texttt{ts}}$	[2, 3, 4, 5]
	$\texttt{tolerance}_{\texttt{ts}}$	[1, 2, 3, 4, 5, 6, 7, 8]

Figure 3-26a shows the results of the sensitivity analysis for each model as a set of Pareto plots. Each point in the plot refers to a specific combination of the nine configuration parameters of the model orchestrator. The prediction accuracy obtained (i.e., MAPE) is represented on the x-axis. In turn, the ratio of trained observations over the total (an indirect measurement of the training time) is represented on the y-axis. The points highlighted in red on each plot represent the Pareto front, the optimal trade-off between prediction accuracy and training time.

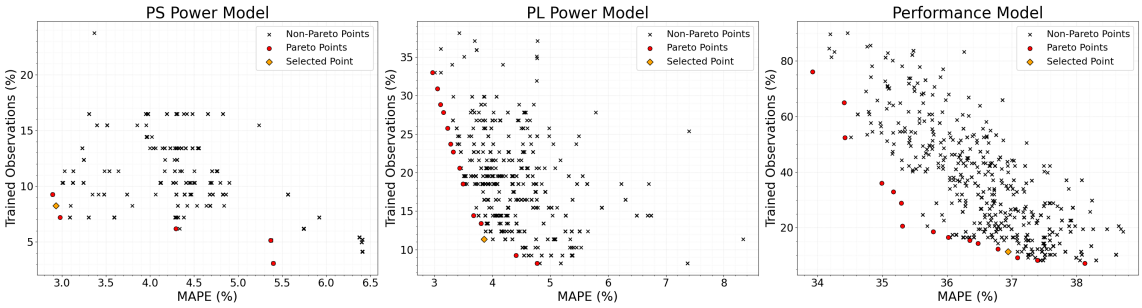
An important caveat of these plots is that points on the Pareto front of one model do not necessarily lie on the Pareto front of the others. The reason is that the modeled variables have a different nature; hence, the same combination of orchestrator parameters affects the training process of each model differently.

The intuitive solution would be to choose a point in the Pareto front of each model, resulting in completely different parameters per model. However, such a strategy cannot be implemented since the model orchestrator trains all models in sync to ensure its execution overhead is kept at minimum values, requiring common parameters (see Table 3-4) to be shared. The solution has been to choose the best combination of common parameters for all models through an extensive analysis of the Pareto plots (the selected values for these parameters are shown on the left-hand side of Table 3-10). Then, a refined grid search is performed to re-evaluate every possible model-specific combination of parameters, maintaining the selected common parameters fixed. These refined Pareto plots are shown in Figure 3-26b.

Since the model-specific parameters evaluated in these graphs are independent between models, the most convenient point of the Pareto front can be selected. The selection criterion has been to reduce computing overhead as much as possible by minimizing training time while maintaining decent prediction accuracy. The selected point for each model is highlighted in Figure 3-26b, and the actual value of each



(a) Sensitivity analysis using every possible combination of parameter values.



(b) Sensitivity analysis using every possible combination of model-specific parameter values.

Figure 3-26: Pareto-based sensitivity analysis of the model orchestrator parameters.

Table 3-10: Selected parameters of the model orchestrator after the sensitivity analysis.

Common Parameters		**Model-Specific Parameters**			
Name	**Value**	**Name**	**Power**		**Performance**
			PS	**PL**	
$\texttt{baseObs}_{\texttt{tr}}$	1000	$\texttt{epochs}_{\texttt{tr}}$	2	2	2
$\texttt{obsRedF}_{\texttt{tr}}$	0	$\texttt{tolerance}_{\texttt{tr}}$	1	3	3
$\texttt{baseObs}_{\texttt{ts}}$	200	$\texttt{epochs}_{\texttt{ts}}$	3	3	3
$\texttt{obsRedF}_{\texttt{ts}}$	0	$\texttt{tolerance}_{\texttt{ts}}$	2	3	6
`idleObs`	2000				

corresponding model-specific parameter is listed on the right-hand side of Table 3-10. The remaining experiments conducted in this chapter use those selected parameters for model orchestration.

Evaluation Scenarios

The following sections evaluate the proposed adaptive learning approach, comparing it with other alternatives in terms of execution overhead and prediction accuracy. Hence, the previously mentioned synthetic workload has been executed using the following four approaches:

- **Baseline:** the hardware acceleration requests of the workload are offloaded to the FPGA with the workload management infrastructure. This approach represents the reference best-case scenario, as it does not involve monitoring nor modeling.
- **Continuous Trace Acquisition:** during the workload offloading, the workload management infrastructure collects periodic power consumption and performance traces. However, no modeling is performed. This approach enables the isolation of the computing overhead induced by the monitoring stage.
- **Continuous Learning:** the workload management infrastructure transfers the monitoring data to the adaptation module. The traces are then processed into observations and used to train power consumption and performance models. This approach learns incrementally in a continuous manner, a strategy followed by previous works found in the literature.
- **Adaptive Learning:** the proposal of this Thesis. In this case, the described resource-aware model orchestrator coordinates the workload management infrastructure and the adaptation module to perform trace monitoring, observation processing, and incremental model updating, aiming to minimize computing overhead.

Please note that the results presented in the following sections are the average of executing a given dynamic workload ten times.

Modeling Accuracy

Figure 3-27 shows the evolution of the prediction accuracy during the execution of the workload under evaluation, comparing the proposed adaptive learning approach (green) with the continuous learning alternative (orange) and a static modeling approach (blue), in which the models are just initially trained, serving as the baseline. The figure represents the MAPE evolution of each modeling approach while modeling the PS power consumption, PL power consumption and execution time of the synthetic workload. Since the execution times vary between approaches due to their associated overheads, the graphs show the number of observations on their x-axes, allowing the generation of a common representation that eases the comparison of the approaches.[16] Furthermore, even though the trace acquisition and model training

[16]This representation of the x-axis is because the figure of merit for this validation section is the prediction accuracy and not the induced overhead, which will be evaluated in the following section.

processes are interleaved in time in the adaptive learning approach (see Figure 3-15), the observations have been evenly distributed to obtain uniform results for the sake of a better and clearer graphical representation.

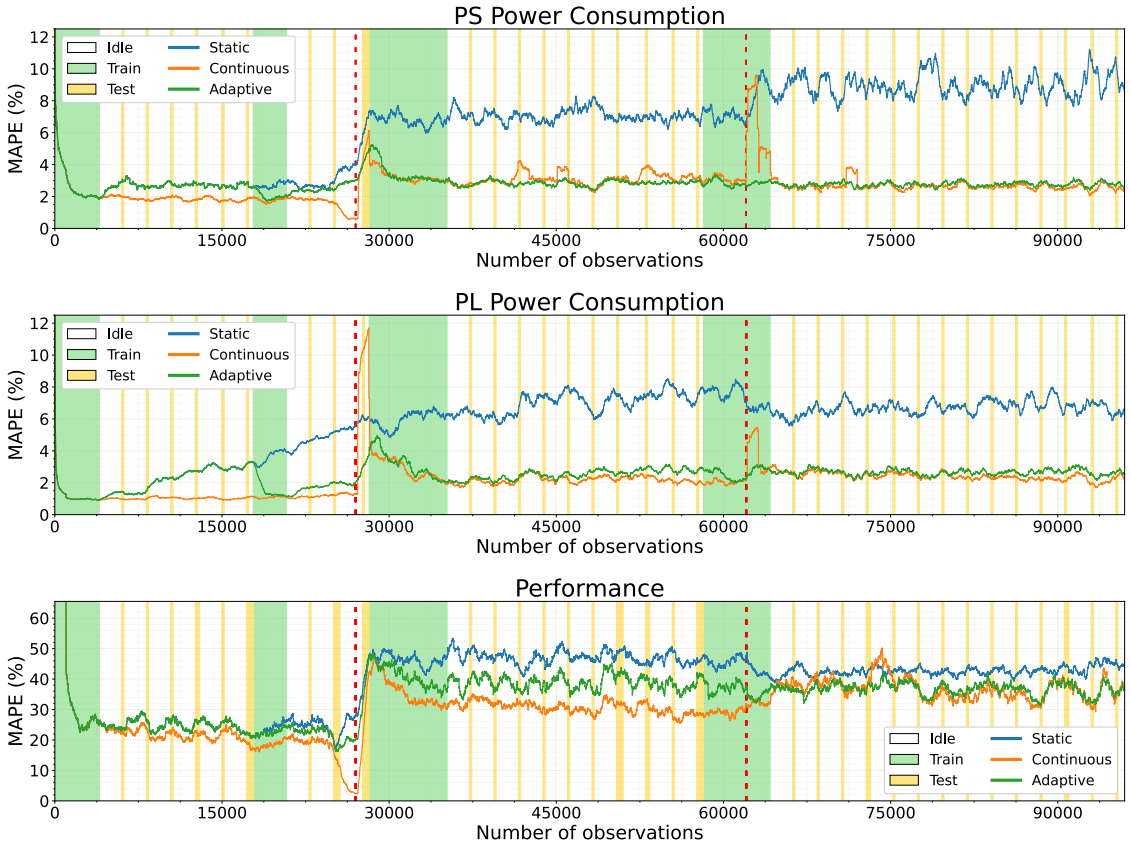


Figure 3-27: Evolution of prediction error for each modeling approach (smaller is better). Red dashed vertical lines indicate when workload changes during the experiments.

The discussion below addresses specific features of the different training processes and considers the behavior of each approach as follows: the static approach trains a model just once in the initial stage (i.e., the first green region in Figure 3-27) with no further model training; the continuous learning approach constantly updates a model on each new observation, collecting traces, processing observations, and training the model non-stop; the adaptive learning approach leverages the resource-aware model orchestrator to update the models incrementally over a series of *Train*, *Test* and *Idle* stages (i.e., green, yellow and white regions in Figure 3-27, respectively).

- **Initial training:** looking at Figure 3-27, it can be observed that the three approaches present the same prediction error evolution during the first training window (i.e., first green region), because training is happening in the three scenarios. Then, the static and adaptive learning approaches, contrary to the continuous learning one, show a degradation in the prediction accuracy since

training is no longer happening. However, in the case of the adaptive learning approach, the model orchestrator periodically inspects the prediction accuracy of the models via the testing stages. In case of a significant deviation between model predictions and actual measurements, the orchestrator can trigger a new training stage for each model (since they are in sync, as described in Section 3.6.2). This scenario is represented in the second training stage (i.e., second green region), particularly for the PL power consumption, where the prediction error of the static approach keeps increasing while the adaptive learning approach starts to catch up with the continuous learning approach.

- **Workload variability:** the behavior of the three approaches also differs in the event of new kernels appearing over time, a phenomenon happening in the cloud-edge continuum scenario that has been simulated by gradually including new kernels to the synthetic workload. Such workload variations are represented by red dashed vertical lines, like the one around the 26,000th observation. Focusing on this first workload change, and after an initial prediction error rise, it can be seen that the continuous learning approach recovers rapidly since it is training the models constantly. Meanwhile, the prediction error of the static approach keeps rising as it executes kernel combinations that include kernels not seen during its initial training stage. On the other hand, the proposed adaptive learning approach initially suffers a similar increase in prediction error since it has not trained the models with these new kernels but, eventually, the model orchestrator, via a testing stage, detects the deviation of the model and starts a new training stage (i.e., third green region). After training a certain number of new observations, the accuracy of the adaptive learning approach becomes comparable to the continuous learning approach.

- **Model drift:** the adaptive learning approach proposed in this Thesis also triggers new incremental learning stages if the prediction error increases significantly (e.g., due to model drift). An example can be observed in the last training stage (i.e., fourth green region), where the training stage starts due to a continued deviation of the prediction error and keeps updating the models until the prediction error becomes acceptable. Please note that, specifically in this case, the adaptive learning approach updates the models with information coming simultaneously from model drift (i.e., what originally triggered the last training stage) and from a workload change (i.e., the inclusion of new kernels, represented in the second red dashed line).

- **Model overfitting:** it can be appreciated that the prediction error of the continuous learning approach presents more sudden changes than the adaptive learning approach, particularly when there are changes in the workload (e.g., around the red dashed lines). The reason is that training using the same kind of observations continuously can produce model overfitting, resulting in more erratic responses to previously unseen kernels, like the ones present in a cloud-edge continuum scenario. In contrast, the model orchestrator of the

adaptive learning approach ceases the training stage as soon as it detects that the prediction accuracy is good enough, minimizing the risk of overfitting.

- **Steady state:** when a change is produced in the workload, it can be observed that the steady state of the prediction error (i.e., stable prediction error value) varies for every approach. This can be better evidenced by looking at the static approach, since it is trained just once for a reduced set of kernels. The reason behind this phenomenon is that the evaluated dynamic scenario presents complex kernel interactions. Hence, introducing new kernels to co-exist with already present ones can produce even more complex kernel interactions that lead to higher prediction errors and larger variability. However, the adaptive learning approach and the continuous learning approach maintain the prediction accuracy of the models fairly stable throughout the workload, as they adapt to dynamic changes.

Modeling Overhead

The execution time of the workload is presented in Figure 3-28 for each described approach, showing its mean and standard deviation normalized to the baseline approach. For a fine-grain analysis, the contribution of the workload manager and the adaptation module has been represented as disjoint contributions.

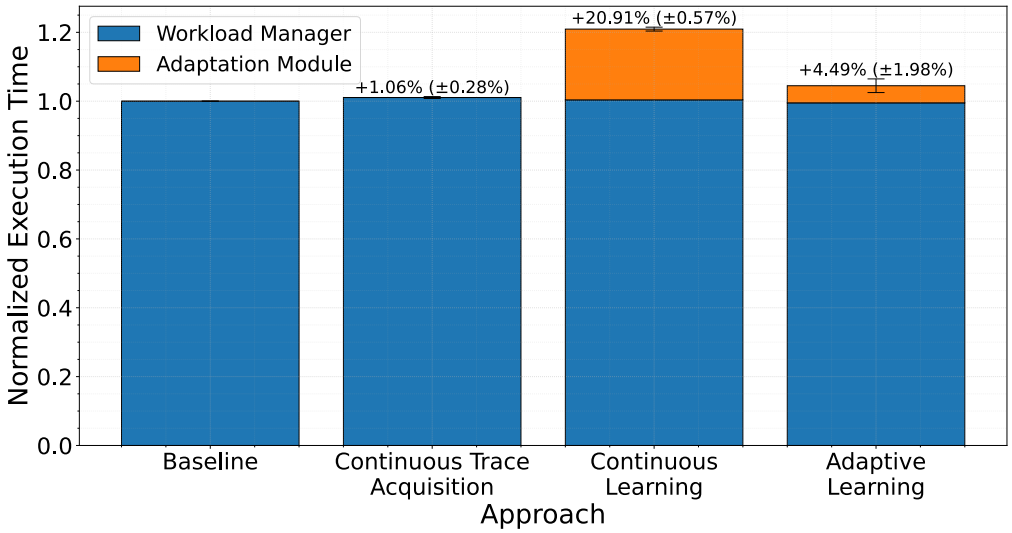


Figure 3-28: Mean and standard deviation of the execution overhead induced by each modeling approach (smaller is better).

The results show a minimal impact of the continuous trace acquisition approach on the execution time of the system. This happens because most of the trace acquisition process is performed by the hardware components of the monitoring framework, which run in parallel to the rest of the system.

Regarding the overhead induced by the continuous learning approach, the figure shows that the continuous processing of the traces and the training models significantly affect the overall execution time of the workload. In fact, an average of around 20% overhead is induced using this approach.

In contrast, the adaptive learning approach proposed in this Thesis obtains an induced overhead 4× smaller than the continuous learning approach, which translates into an overhead below 5% compared to the baseline approach, while keeping the models up-to-date. This lower overhead comes from the ability of the adaptive learning approach to reduce the training time via the model orchestrator, which has a significant impact on the overhead, as in the selected ML algorithms (i.e., regression trees), training is more expensive than predicting.

It is also relevant to consider the overhead variability of each approach, for which the proposed adaptive learning approach has the highest. Since the other approaches perform the same procedures constantly during their operation, each of them experiences a fairly consistent execution time when running the workload. However, the proposed adaptive learning approach adapts the learning process at run time. Therefore, due to the dynamic nature of the overall system, training or test phases might vary slightly, rendering higher overhead variability. Nevertheless, the measured standard deviation is below 2% of the total execution time.

Analysis and Discussion

The proposed adaptive learning approach, as shown in Figure 3-27, has demonstrated a prediction error evolution per model similar to the continuous learning approach, sometimes even better. The model orchestrator has been able to quickly detect degradation in the prediction accuracy below acceptable levels and update the models to keep them in good shape. In addition, it has been shown to minimize overfitting, mitigating the impact of unseen observations on the prediction accuracy of the models and maintaining a stable prediction accuracy over time. But above all, as demonstrated in Figure 3-28, the proposed approach accomplishes equivalent prediction accuracy while inducing a quarter of what the continuous learning counterpart induces in the system in terms of computing overhead.

A summary of the results is shown in Table 3-11, which presents a comparison between each modeling approach in terms of prediction error and induced computing overhead. Concerning overhead, workload execution time increases in both continuous learning and adaptive learning approaches compared to the static approach, which presents no training overhead. To evaluate prediction accuracy, the MAPE value is presented for each predicted variable, using the continuous learning approach as the reference (since it consistently gives the best results), while the MAPE of the other methods is represented as the difference from the reference (in absolute terms). Please note that the values shown in this table are computed as the average of ten workload executions performed with each modeling approach. Power consumption is a simple additive variable. In turn, execution time (i.e.,

computing performance) is a complex variable strongly affected by the specific nature of each kernel and the complex interactions between them that arise when running the dynamic workload of the cloud-edge continuum. This explains the significant variation in prediction accuracy when modeling each target variable.

Table 3-11: Induced overhead and prediction error per modeling approach (smaller is better).

Modeling Approach	Overhead (%)	Model Error - MAPE (%)		
		PS Power	PL Power	Performance
Static	–	6.56 (+3.86)	5.73 (+3.61)	39.39 (+8.97)
Continuous	20.91	2.70 (–)	2.12 (–)	30.42 (–)
Adaptive	4.49	2.78 (+0.08)	2.46 (+0.34)	34.20 (+3.78)

The adaptive learning approach proposed in this Thesis renders a prediction error comparable to that of the continuous learning approach, increasing the MAPE in less than 1% when predicting power consumption and less than 4% when predicting performance. Furthermore, the overhead induced by the adaptive learning approach is four times smaller than the overhead caused by the continuous learning approach. It should be noted that although the MAPE of the static approach may not appear to be significantly higher than the other approaches, its accuracy deteriorates over time due to changes in workload, operating conditions, or the system itself. In contrast, the proposed adaptive learning approach maintains the prediction accuracy stable under such variations.

3.7.3 Portability and Generalizability of the Modeling Methodology

There is a diverse set of reconfigurable platforms that might be used in the cloud-edge continuum. Therefore, portability and generalizability are essential aspects of any infrastructure or methodology designed to be part of such a scenario. It is so in every solution proposed in this Thesis (e.g., the workload management infrastructure, the monitoring framework, the modeling strategy), which are built to be configurable and modular, behind the design principles of portability and generalizability.

This section extends the evaluation in terms of portability and generalizability to the proposed modeling methodology. To this end, the exact same experiments described in the previous section for the ZYNQ UltraScale+ ZCU102 platform have been conducted again in this section on: i) a Pynq-Z1 development board, a lower-end platform with a PS frequency of 650 MHz and a PL frequency of 100 MHz,[17] and ii) an Alveo U250 accelerator card, a cloud-grade FPGA configured at 100 MHz and attached to an Intel Xeon Gold 6338 host CPU via PCIe.[18] In the Pynq-Z1 setup, the

[17] XC7Z020-1CLG400C device.
[18] A-U250-P64G-PQ-G device.

maximum number of parallel accelerators is limited to 4 due to the restricted amount of logic resources available in the device, while the Alveo U250 setup features 16 parallel accelerators exploiting its vast resources. Moreover, the only way to measure power consumption from these new platforms is directly from their power supply, obtaining the power consumption of the entire platform as a single power rail. To ease evaluation, Table 3-12 and Table 3-13 present a side-by-side comparison between the results obtained in the previous section (identified as *Z* for the ZCU102 platform) and the results obtained with these new platforms (identified as *P* for the Pynq-Z1 platform, and *A* for the Alveo U250 platform). The following sections will discuss the portability and generalizability capabilities of the proposed solutions to the new targets.

Table 3-12: Comparison of the induced overhead per modeling approach on every platform.

Modeling Approach	Overhead (%)		
	Z	**P**	**A**
Static	–	–	–
Continuous	20.91	24.64	0.85
Adaptive	4.49	6.07	0.72

Z: Stands for the Zynq UltraScale+ ZCU102 board
P: Stands for the Pynq-Z1 board
A: Stands for the Alveo U250 board

Table 3-13: Accuracy comparison per modeling approach and platform (smaller is better).

Modeling Approach	Model Error - MAPE (%)						
	PS Power	**PL Power**	**Board Power**	**Board Power**	**Performance**		
	Z	**Z**	**P**	**A**	**Z**	**P**	**A**
Static	6.56 (+3.86)	5.73 (+3.61)	3.16 (+2.43)	1.87 (-4.54)	39.39 (+8.97)	38.85 (+20.71)	31.05 (+6.99)
Continuous	2.70 (–)	2.12 (–)	0.73 (–)	6.41 (–)	30.42 (–)	18.14 (–)	24.06 (–)
Adaptive	2.78 (+0.08)	2.46 (+0.34)	1.06 (+0.33)	0.63 (-5.78)	34.20 (+3.78)	20.17 (+2.03)	18.93 (-5.13)

Z: Stands for the Zynq UltraScale+ ZCU102 board (i.e., main experiments)
P: Stands for the Pynq-Z1 board (i.e., portability experiments)
A: Stands for the Alveo U250 board (i.e., portability experiments)

Infrastructure Portability

Migrating the workload management infrastructure and the adaptation module to different platforms is a straightforward process. Hardware elements are configured for the desired FPGA target as part of the build process, and software components are inherently compatible with various CPU architectures without requiring changes.

The primary consideration is the variation in the number of power rails on the target platforms (in both cases, a reduction from two to one). Nevertheless, both hardware and software components have been developed with this in mind and support specific configuration options to adapt to platform-specific constraints. As a result, when the number of monitored power rails changes, the data acquisition and processing flow is adjusted accordingly. The same applies to the number of models created and the model orchestration mechanism. Furthermore, if any custom processing is required for the new power rail, the adaptation module is designed to allow the addition of new processing routines with minimal effort.

In this scenario, where only a single power rail reflects the total power consumption of the system, the hardware samples power values from just one ADC channel rather than two. Consequently, the software processes a single set of power values and builds a corresponding power model.

Regarding the overhead measurements shown in Table 3-12, for the Pynq-Z1 scenario, the overhead remains comparable to the main experiments, confirming that the proposed infrastructure operates consistently even on a new target platform with a different CPU architecture and reduced computational capabilities. In contrast, for the Alveo U250 scenario, the modeling process has a negligible impact on both the continuous and adaptive approaches. This outcome is expected, as the selected ML algorithms were deliberately chosen to be as lightweight as possible to minimize the overhead on low-end devices; consequently, a high-end device, such as the host CPU used in the Alveo U250 scenario, experiences minimal overhead. Nevertheless, the adaptive approach still yields almost three times less overhead than the continuous approach, indicating that on such devices, the complexity of the models could be increased while still benefiting from the lower overhead of the adaptive approach compared to its continuous counterpart.

Modeling Consistency

To assess the generalizability of the proposed modeling methodology, the same models and configuration settings from the previous section have been used during these portability experiments. It is important to highlight that these new target platforms differ significantly from the main experiments. Let us treat both the Pynq-Z1 and the Alveo U250 scenarios separately.

The Pynq-Z1 platform presents less computational power, fewer reconfigurable slots, a different reconfiguration workflow and a fundamentally new power consumption nature, as the entire power consumption of the platform is represented

by a single rail. When observing the prediction error values in Table 3-13, it is observed that the models achieve, in general, better accuracy than in the previous tests. This improvement is likely due to the reduced number of reconfigurable slots, which simplifies the prediction of kernel interactions. Nevertheless, the difference in prediction accuracy between the adaptive and continuous learning approaches (values shown in parentheses) remains consistent between both platforms, meaning that the modeling methodology performs similarly in both cases. It is worth mentioning that the adaptive learning approach still minimizes training overhead while preserving prediction accuracy.

In the Alveo U250 scenario, the platform provides greater computing power, a higher number of reconfigurable slots, and, similar to the Pynq-Z1 case, a different reconfiguration workflow and a distinct power consumption profile (full-board measurements). Intuitively, one might expect the opposite trend to that observed on the Pynq-Z1, anticipating lower accuracy due to the increased complexity of kernel interactions. However, the results show that prediction accuracy is slightly higher.

For power prediction, this outcome is explained by the fact that, as shown in Section 2.5.2, the Alveo U250 power consumption is not significantly affected by hardware execution. In such conditions, the adaptive model achieves the best results, as it only trains when conditions change, keeping the models properly updated. Meanwhile, further retraining on new samples that might not substantially affect the power consumption would only introduce noise and degrade accuracy. This explains why even the static model, which is only trained once initially, has even better results than the continuous learning approach.

Performance prediction requires a more careful analysis. While it is reasonable to expect that more reconfigurable slots would lead to more complex interaction patterns and thus lower accuracy, the results indicate otherwise. In the Alveo U250 case, 16 accelerators execute a set of 11 kernels, whereas in the ZCU102 case, 8 accelerators execute the same 11 kernels. On the ZCU102, many distinct subsets of up to 8 kernels can be configured, producing a broad range of scenarios. On the Alveo U250, the highest concurrency level involves the 11 kernels, and only one such combination exists. This reduces the diversity of high-contention configurations, making the relationship between features and performance more consistent and easier to learn.

While all modeling approaches benefit from the reduced diversity of interaction patterns on the Alveo U250, the adaptive learning strategy achieves the highest performance prediction accuracy. In this scenario, the lower complexity of kernel interactions increases the risk of overfitting for the continuous learning approach, as it continues to train on similar observations over time. When occasional variations in execution behavior occur, such as changes in kernel combinations, an overfitted model is less able to provide accurate predictions, resulting in noticeable accuracy drops. The adaptive approach mitigates this by stopping the training stage once the prediction accuracy reaches an acceptable level, preserving its ability to generalize to these less frequent operating conditions. Consequently, the adaptive model delivers more stable and accurate predictions in the Alveo U250 scenario.

Sensitivity Analysis

Please note that the configuration parameters of the model orchestrator remained unchanged from previous experiments. Despite the nature of the predicted variables having changed significantly on the new platforms (e.g., different kernel interaction, different CPU contention, completely different power consumption behavior), the modeling results are even superior. This can be attributed to using MachSuite as the benchmark for the sensitivity analysis, since it includes a wide range of computation/communication patterns common to all target applications the system might come across, despite any platform differences. Hence, the selected configuration parameters for the model orchestrator appear to generalize well across platforms rather than being tied to specific platform features.

These results show that porting the proposed solution to different platforms is practical and manageable, requiring only some minor changes. The performance of the system and the modeling accuracy remain stable, proving the high level of generalizability and portability achieved with the proposed solutions.

3.7.4 Conclusions

This chapter has presented a methodology for characterizing dynamic workloads at run time in reconfigurable multi-accelerator systems. The impact of kernel interactions on the execution of dynamic workloads has been evaluated. The feasibility of characterizing these interactions has been demonstrated through an offline workload characterization approach, which has then been extended with incremental ML-based models and a dedicated model orchestration mechanism, targeting run-time workload characterization with minimal overhead.

The validation campaigns presented in this chapter provide two key insights that highlight the contributions and practical value of the proposed solutions of this Thesis.

First, the implementation of the proposed modeling methodology and the integration of incremental machine learning techniques enable accurate prediction of power consumption and performance in reconfigurable multi-accelerator systems operating under dynamic workloads. The proposed adaptation module processes run-time traces to update models incrementally, while the resource-aware orchestrator effectively manages the learning process to minimize system impact. This strategy achieves prediction accuracies comparable to continuous learning approaches, with less than 4% increase in prediction error, while introducing only a 4.49% execution overhead (i.e., more than four times lower than the 20.91% overhead induced by the continuous learning approach). Moreover, the modeling infrastructure is capable of capturing highly complex kernel interactions, which can cause execution time variations of up to 500%, achieving, in a dedicated test, a mean relative error below 0.6% in predicting these effects. This level of accuracy in such a dynamic context supports the strength of the proposed modeling methodology and its suitability for a cloud-edge continuum scenario.

Second, the results demonstrate that the entire infrastructure (i.e., monitoring framework, workload management infrastructure, modeling strategy) is both portable and generalizable. The proposed solutions adapt seamlessly to platforms with distinct architectures, reconfiguration mechanisms, and power profiles. The experiments carried out on different platforms confirm that the prediction accuracy and system overhead remain stable across platforms, without requiring a fine-tuning of the configuration parameters. This robustness confirms that transferring the proposed solutions to new environments is feasible, maintaining their effectiveness, which is essential for solutions to be deployed at the cloud-edge continuum.

Chapter

4 SCHEDULING FOR OPTIMAL RESOURCE MANAGEMENT

This chapter proposes a workload optimization methodology that relies on a metaheuristic-based task scheduling algorithm to perform run-time optimization of dynamic workload deployments in reconfigurable multi-accelerator systems. The methodology leverages the run-time characterization described in the previous chapter to predict task interaction and guide the scheduling process. A multi-objective workload optimization approach is adopted to improve workload makespan (as well as energy efficiency, indirectly) and fair use of resources in the long run.

An overview of the components that form the proposed workload optimization methodology is presented first, followed by an exploration of the state-of-the-art alternatives found in the literature. Subsequently, an in-depth description of the proposed scheduling methodology implementation is presented. The chapter ends with a validation campaign to evaluate the proposal.

4.1 Chapter Overview

Although reconfigurable FPGAs offer significant potential within the cloud-edge continuum, exploiting their full benefits in such contexts requires addressing a fundamental and complex issue: in a multi-accelerator architecture like the target of this Thesis, it must be decided which acceleration request from the workload to execute and when to do it (i.e., scheduling). Moreover, it must also be determined where to deploy the acceleration request among the available resources and the number of parallel replicas (i.e., allocation) [Diessel'01].[1] This scheduling problem falls into the NP-hard category [Blazewicz'83] and belongs to the broader class of combinatorial optimization problems, whose goal is to identify the best solution from an immense pool of possibilities [Adhi'18]. In slot-based acceleration systems like the one adopted in this Thesis, the number of potential configurations expands exponentially with the number of tasks and the number of available reconfigurable slots, making brute-force methods impractical, especially in performance-critical environments with latency constraints.

[1] Both scheduling and allocation are the result of the same procedure in this work; hence, this Thesis utilizes both terms interchangeably.

To mitigate these difficulties, most existing research works rely on offline profiling of workloads to inform scheduling and placement [Tianyang'21]. In such approaches, task information (e.g., duration, energy consumption, and communication costs) is characterized in advance under static and controlled scenarios. This pre-collected data is then used either during the design process to create static scheduling plans or at run time to guide optimization routines (e.g., heuristic strategies) that determine task allocation. However, these methods are inherently designed for fixed workloads that are entirely known in advance.

In contrast, dynamic systems such as those in the cloud-edge continuum operate under conditions where workloads are unpredictable at design time and subject to frequent changes (see Section 1.4.2). Under these conditions, tasks arrive irregularly, behave differently due to input variability or environmental conditions, and target diverse heterogeneous computing platforms. Here, prior profiling is impractical and may also decrease performance by ignoring the dynamic nature of run-time task interactions, such as resource contention between concurrent accelerators.

To overcome these issues, this Thesis introduces a workload optimization methodology tailored for reconfigurable systems with multiple accelerators, aiming to minimize makespan (indirectly improving energy efficiency) and fair use of resources. Previous results indicated that concurrent execution of conflicting tasks could lead to performance degradation of up to 500%, attributed to shared resource contention (see Section 3.3). Based on this, a conflict-aware optimization strategy that takes into account the data-driven workload characterization models developed in Chapter 3 is proposed. This approach effectively enables dynamic profiling at run time, allowing the system to adapt its behavior on the fly to changing workloads and conditions without requiring prior assumptions or manual configuration, achieving a self-adaptive, autonomous optimization.

To efficiently explore the extensive scheduling solution space, the proposed optimization strategy relies on the Crow Search Algorithm (CSA) metaheuristic [Askarzadeh'16], diverging from other traditional heuristic-based methods found in alternative state-of-the-art approaches. In this Thesis, metaheuristics are selected for their ability to provide generic, high-level strategies that can tackle complex problems and yield broadly applicable solutions [Yang'10, Houssein'21]. This makes them particularly suitable for cloud-edge continuum systems, where portability and adaptability across diverse platforms are essential.

Figure 4-1 depicts the main components of the proposed workload optimization methodology, highlighted with respect to the rest of the components of this Thesis. The workload optimization process is as follows: every time a new task from the workload arrives or a reconfigurable slot becomes available, the conflict-aware scheduler gathers run-time information regarding the tasks waiting in the queue and the tasks running in the FPGA fabric. Using that information, it starts the optimization process, where the CSA metaheuristic leverages run-time ML-based estimations (updated using power and performance predictions from the incremental models) to search the scheduling solution space, evaluating a subset of scheduling configuration candidates. When the

optimization process is completed, the scheduler selects the best scheduling candidate (i.e., a set of tasks and the number of replicas) based on a multi-objective fitness function. The scheduling decision is then configured in the FPGA. The scheduler promotes scheduling candidates that minimize the loss of performance due to the conflict between tasks (i.e., task interaction), offering a multi-objective long-term workload optimization, either targeting makespan or fair use of resources. This workload optimization process is repeated in a loop during the workload execution.[2]

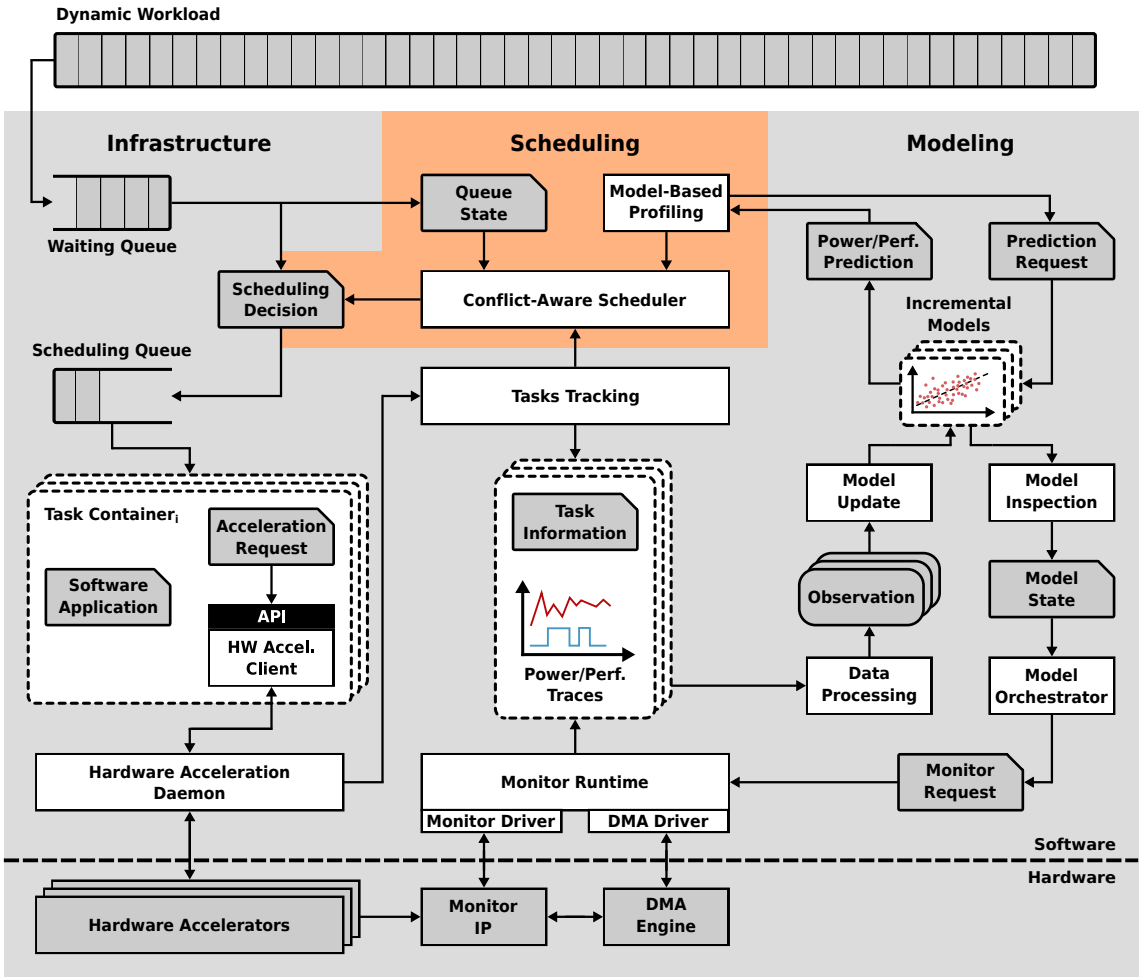


Figure 4-1: Details of the scheduling strategy proposed in this Thesis.

This workload optimization methodology is open source and available online.[3]

Original contribution 4-1 *A metaheuristic-based strategy that relies on data-driven predictive models to schedule dynamic workloads in reconfigurable multi-accelerator systems, optimizing and trading off energy and performance on the one hand, and fair use of resources on the other hand, at run time.*

[2]The scheduler makes scheduling decisions only when the incremental models are already trained; during their training phase, the task scheduling follows an FCFS approach.

[3]https://github.com/des-cei/fpga-scheduling/

4.2 State of the Art

To tackle the problem of workload optimization in FPGAs, the research community has traditionally adopted a well-established methodology: conduct a design-time characterization of the tasks intended for acceleration on the FPGA and then utilize this information at run time to guide scheduling decisions [Tianyang'21]. The objective of such scheduling strategies is to improve workload execution (mainly for Directed Acyclic Graph (DAG)-structured applications whose characteristics are fully known during design time) either by reducing makespan [Ramezani'20, Bertolino'20, Alismail'23, Khetawat'24, Vaishnav'19, Duhamel'22, Vaishnav'20], lowering power consumption [Paul'24], or promoting fair resource usage [Mehrabi'22, Karabulut'25].

Although the dominant focus in the literature is on makespan minimization, the specific implementation approaches vary slightly across studies. As an example, the authors in [Ramezani'20] introduce a method based on reconfiguration prefetching, aiming to overlap the reconfiguration phase of a future task with the execution time of an ongoing one [Li'02]. Their solution extends this principle to multiple DAGs, fetching tasks from upcoming graphs while executing the current one. While this strategy assumes advanced knowledge of several DAGs, their findings demonstrate a nearly 22% reduction in overall makespan. An analogous principle is applied in [Bertolino'20], where the task latency information of a DAG is used to classify tasks into distinct subgroups. Each subgroup includes at least one task with notably higher latency than the others. As a result, when these groups are deployed together on the FPGA, the execution of the longest task effectively hides the duration of the shorter ones. In contrast, [Alismail'23] exploits memory-related task metrics, such as bandwidth usage and memory access count, to pair tasks with complementary memory profiles. The results show that running such combinations in parallel reduces contention, improving over their prior work by as much as 65%.

Other approaches place greater emphasis on dynamically deciding whether to offload tasks to the FPGA or run them as software, a consideration especially relevant in heterogeneous platforms. This is seen in [Khetawat'24], where the authors choose to assign specific tasks to the CPU or GPU when their speedup on the FPGA is less advantageous. Their decision mechanism combines pre-characterized profiles with run-time resource monitoring, enabling efficient allocation of computing resources across a workload and delivering a 1.25× performance gain. A related heuristic is found in [Vaishnav'19], where the system selects both the processing unit (e.g., FPGA or CPU) and the number of parallel task instances, relying again on static profiling data. This strategy achieves a 20% makespan reduction compared to baseline methods.

Performance improvements have also been tackled by exploring different variations in task configuration. In the approach taken by [Duhamel'22], a design-time exploration process generates multiple versions of each task, each with different performance and QoS trade-offs. At run time, if an application deadline is at risk, the system opts for a lower-QoS variant (i.e., reduced video quality in multimedia

workloads) to favor faster execution. This philosophy is mirrored in [Vaishnav'20], where different parallelism levels are assigned to each task depending on system conditions. Their goal is to avoid task starvation while maintaining high throughput. High-parallelism configurations are preferred when sufficient resources are available, but are replaced with lighter alternatives when additional tasks must be executed. Again, this relies on prior knowledge of all task variants and their characteristics.

The topic of power efficiency is also addressed in [Paul'24], where profiling results are used to determine combinations of tasks that, when running in parallel, achieve maximal per-task parallelism with minimal overall power draw. These combinations are prioritized at run time to improve energy efficiency without sacrificing throughput.

In parallel, multiple studies explore heuristics that promote fair resource usage at run time. A representative case is [Mehrabi'22], where the proposed heuristic keeps track of the FPGA resource usage demanded by each task. Based on this metric, tasks are dynamically assigned a priority that is inversely proportional to their accumulated usage, encouraging fairer access to the hardware. These priorities are updated on the fly, and the results indicate increased FPGA utilization and improved performance relative to Round-Robin (RR) strategies. A similar policy is proposed in [Karabulut'25], in this case, assigning priorities to tenants rather than tasks. The fairness-driven scheduling again leads to more balanced resource distribution and higher utilization.

Whether the aim is to enhance performance or reduce power consumption, it is clear that most scheduling techniques for FPGAs depend largely on design-time analysis. This information is then reused during execution to drive optimizations. While effective in static contexts, such strategies overlook certain aspects that occur at run time, such as task interactions during concurrent execution. For scenarios involving dynamic workloads, such as those associated with the cloud-edge continuum addressed in this work, incorporating run-time adaptability into the scheduler becomes not just beneficial but necessary.

The management of resources outside the field of FPGAs follows a similar trend: a design-time profiling leveraged at run time to perform optimizations. However, as highlighted in the survey [Goodarzy'20], cloud computing relies on more advanced techniques for offline profiling (e.g., supervised learning) compared to earlier ad-hoc implementations. These methods enable run-time estimations that provide a limited degree of adaptivity in their scheduling strategies. For instance, the authors in [Smith'24] propose a thermal-aware scheduling for GPUs at data centers. Their solution trains a supervised learning model at design time to predict the temperature of a GPU when running different tasks. Then, at run time, while the workload is executed, temperature predictions are made to take scheduling decisions that homogenize the temperature levels of a set of GPUs, rendering 12.82% energy savings compared to non-thermal-aware solutions. There are also similar solutions that target performance optimizations. An example of such is proposed in [Zacarias'21], where the authors make workload scheduling decisions in HPC scenarios to minimize performance degradation when running tasks in parallel. At design time, the tasks are executed to collect performance data using performance counters. This information

is then fed to a supervised learning model, which is trained to predict the performance degradation of the co-located tasks. At run time, these predictions are leveraged to select task combinations with minimal performance degradation as schedule candidates. The approach results in up to a 12% performance improvement compared to traditional scheduling policies. These approaches, while presenting a more refined profiling, still require design-time knowledge, which is unfeasible for the dynamic workloads addressed in the cloud-edge continuum scenario.

Unlike traditional approaches found in the literature, this Thesis targets the optimization of FPGAs performance under dynamic workload conditions. Rather than relying on design-time profiling, which demands extensive prior insight into the workload, the proposed approach introduces ML-based models capable of characterizing workloads at run time. These models provide real-time insights that are leveraged directly to inform the optimization of task scheduling. The proposed scheduler explicitly addresses the reduction of kernel interactions associated with the use of shared resources in the FPGA fabric (such as memories, buses, or interfaces, among others), a factor ignored in previous research works despite its substantial influence on overall system performance, as discussed in Section 3.3. In addition, this work adopts a metaheuristic algorithm instead of a conventional heuristic. Metaheuristics offer broader applicability across different problem domains, making them more adaptable to heterogeneous and evolving environments like those in the cloud-edge continuum [Houssein'21]. A qualitative assessment comparing existing methods and the approach developed in this Thesis is summarized in Table 4-1.

Table 4-1: State of the Art - Task scheduling approaches for workload optimization.

Reference	Platform	Domain	Optimization	Profiling	Workload
[Ramezani'20]	FPGA	E	Makespan	Design Time	Static
[Bertolino'20]	FPGA	C	Makespan	Design Time	Static
[Alismail'23]	FPGA	E	Makespan	Design Time	Static
[Khetawat'24]	Heterogeneous	C	Makespan	Design Time	Static
[Vaishnav'19]	Heterogeneous	E	Makespan	Design Time	Static
[Duhamel'22]	FPGA	E	Makespan	Design Time	Static
[Vaishnav'20]	FPGA	E	Makespan	Design Time	Static
[Paul'24]	FPGA	C	Power	Design Time	Static
[Mehrabi'22]	FPGA	E	Fairness	–	Static/Dynamic
[Karabulut'25]	FPGA	C	Fairness	–	Static/Dynamic
[Smith'24]	GPU	C	Power	Design Time	Static
[Zacarias'21]	CPU	C	Makespan	Design Time	Static
This work	FPGA	C/F/E	Makespan/Fairness	Run Time	Static/Dynamic

N/A (–) | **OR** (/)

Domain
C: Cloud computing
F: Fog computing
E: Edge computing

Optimization
Makespan: The workload execution time and, indirectly, the energy consumption
Power: Specifically optimizing power consumption
Fairness: Foster a balanced resource distribution

4.3 Conflict-Aware Scheduler

This section provides a detailed description of the workload optimization approach presented in this Thesis. To provide an in-depth explanation of the proposed optimization mechanism, the dedicated subsections deal with different relevant aspects of the proposal. First, the optimization problem to be solved is described, followed by a conceptual explanation of the proposed conflict-aware solution. Then, the actual implementation of the scheduler is introduced, together with an enumeration of the modifications required for adapting it to the target problem. Finally, the selected fitness function is presented.

4.3.1 Presentation of the Optimization Problem

The core challenge of this chapter is to efficiently schedule hardware acceleration requests on an FPGA. The proposed scheduler is designed to operate at run time, dynamically selecting the most appropriate combination of tasks to offload to the FPGA at each time. This decision-making process is carried out in response to the arrival of new tasks or changes in resource availability. Within the slot-based reconfigurable architecture adopted in this Thesis, this involves defining the specific configuration of each slot by selecting which tasks to execute in parallel on the FPGA, as well as how many replicas of each, thus leveraging both data-level and task-level parallelism through the use of DPR.

A key assumption in this context is that task execution on the FPGA is non-preemptive, meaning that once a task begins execution, it runs to completion before any replacement occurs. This is due to the prohibitive overhead generally associated with preemption in reconfigurable FPGA-based systems. As such, each scheduling decision must consider both the tasks that are currently running and the ones waiting for execution, selecting from the waiting queue those that are expected to provide the most beneficial long-term outcome. This decision-making cycle is repeated whenever a new task arrives or when reconfigurable slots become available in the FPGA fabric.

Figure 4-2 shows an illustration of this scheduling problem. Observe that the scheduler must account for running tasks (i.e., `A`) and choose a configuration candidate among the possible task combinations from the waiting queue (i.e., `B` and `C`). In this case, the solutions considered are those that keep task `A` running where it has been decided in a previous evaluation.

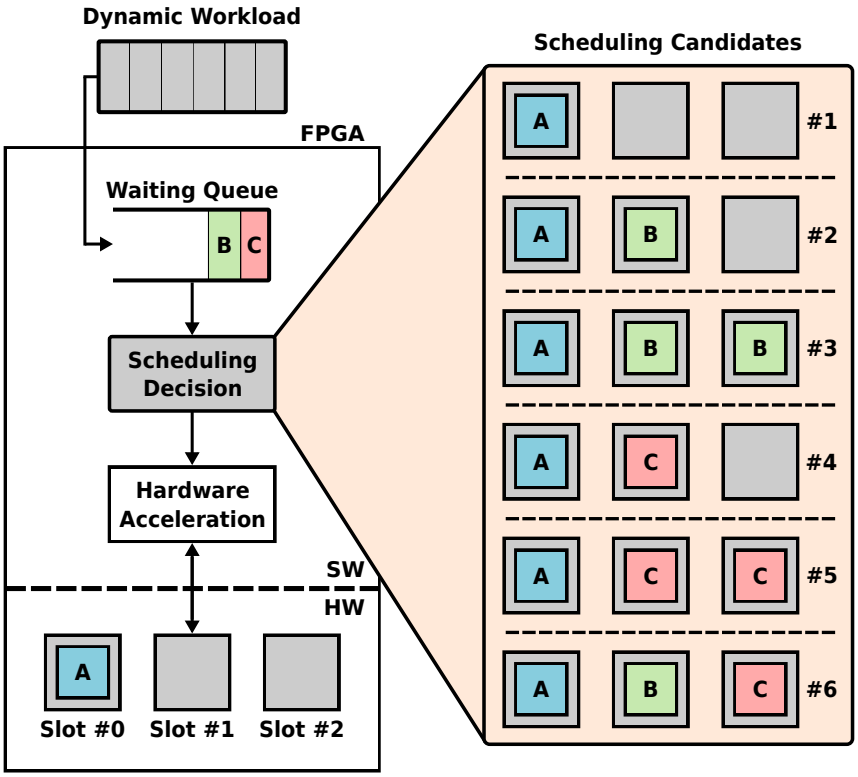


Figure 4-2: Illustration of the scheduling problem in slot-based FPGAs.

4.3.2 Proposed Solution

To determine the most suitable configuration, this Thesis proposes leveraging the run-time workload characterization mechanisms from Chapter 3 to assess each scheduling candidate (i.e., possible configuration). Specifically, the method employs the predictive models to estimate the level of interaction between tasks running in parallel (typically caused by resource contention), which can significantly degrade performance, as analyzed in Section 3.3. Based on these predictions, the scheduler selects the configuration that minimizes such interaction, aiming to reduce the overall makespan. The underlying rationale is that consistently choosing configurations with minimal task interaction will yield more efficient workload executions over time.

Nevertheless, the number of potential configurations grows rapidly with the number of available slots and pending tasks, making exhaustive or greedy search strategies computationally unfeasible in some scenarios [Adhi'18]. To address this, the proposed approach adopts a metaheuristic optimization algorithm to intelligently explore the solution space, evaluating only a subset of promising candidates. Among the different categories of metaheuristics, such as evolutionary algorithms, physics-inspired methods, and population-based strategies, the latter has shown particularly effective results in complex scheduling scenarios like the one considered in this Thesis [Aron'22].

The metaheuristic chosen for this scheduler is the CSA [Askarzadeh'16], which has proven effective across a wide range of domains and offers competitive performance compared to well-known alternatives like Particle Swarm Optimization (PSO) [Kennedy'95] and Ant Colony Optimization (ACO) [Dorigo'06]. An additional advantage of CSA, which favors its use over other alternatives in this Thesis, is its relatively lightweight and straightforward implementation [Singh'21], which is especially beneficial in the target scenario, where multiple computational processes (i.e., run-time scheduling, dynamic modeling, and hardware task deployment) are executed concurrently on the same platform and overheads need to remain minimal. Nevertheless, the proposed optimization methodology is not dependent on the selected scheduling algorithm; therefore, employing any other alternative would not require major architectural changes.

4.3.3 Scheduler Implementation: The CSA Algorithm

The CSA algorithm draws inspiration from the natural behavior of crows [Askarzadeh'16]. These birds are known for their intelligence, particularly their ability to store food in specific locations and recall these spots over time. Additionally, crows observe the actions of others, which enables them to: i) track fellow crows in an attempt to uncover and steal hidden food, and ii) become aware of being pursued and subsequently flee to a random location to keep their hidden food secret. A notable aspect of this algorithm's implementation is its simplicity, as it depends on just two configurable parameters: Flight Length (fl), which defines how far a crow can move in one step, and Awareness Probability (AP), the probability that a crow notices it is being followed. These two variables govern the algorithm's balance between exploration and exploitation.

From a conceptual point of view, these two behaviors of the algorithm work as follows. During each iteration, crows determine whether or not to follow another individual, a decision influenced by the AP parameter. If a crow chooses not to follow, it explores a new, random location (representing exploration). Conversely, if it decides to follow, it travels near (or directly to) the target crow's food hiding spot, exploiting previously successful areas (representing exploitation).

Algorithm 4-1 shows the pseudocode of the CSA, and a detailed description of its operation is presented next:

1. The initial positions of the crows are randomly assigned within a d-dimensional search space, where d represents the number of variables to optimize (line 1). In our case, there will be a variable for each task to be scheduled.

2. Each crow's starting location is evaluated and stored as its initial memory (line 2). This memory, denoted as m_i, retains the crow's best-known food hiding spot. Note that only the best position encountered by each crow is saved in its memory.

3. In order to modify its position, a crow x_i randomly selects another crow x_j to follow (line 5). A random number r_i is then generated. If this value exceeds the

Algorithm 4-1 CSA: Crow Search Algorithm

Input: n Number of crows in the population
$\text{iter}_{\max}$ Maximum number of iterations
Output: Optimal crow position
1: Initialize positions of crows (d-dimensional space)
2: Initialize crows' memory m_i
3: **for** iter = 1 to $\text{iter}_{\max}$ **do**
4: **for** each crow x_i in the population **do**
5: Choose a random crow x_j
6: Determine the awareness probability AP
7: Update $x_{i,\text{iter}+1}$ using Equation 4-1
8: Check solution boundaries
9: Calculate the fitness of each crow
10: Update crows' memory using Equation 4-2

awareness probability (AP), the crow x_i attempts to reach the food hiding spot m_j of crow x_j (line 6).

The crow's updated position x_i is computed as follows (line 7):

$$x_{i,\text{iter}+1} = \begin{cases} x_{i,\text{iter}} + r_i \times fl_{i,\text{iter}} \times (m_{j,\text{iter}} - x_{i,\text{iter}}) & \text{if } r_j \geq AP_{j,\text{iter}} \\ \\ \text{a randomly selected location} & \text{otherwise} \end{cases} \tag{4-1}$$

where $iter$ indicates the current iteration count; $AP_{j,\text{iter}}$ is the awareness probability for crow x_j; r_j and r_i are independent random numbers that influence whether a crow follows another and how far it flies, respectively; and $fl_{i,\text{iter}}$ stands for the flight length used by crow x_i during its attempt to reach m_j.

4. The new position is validated to ensure it remains within the defined d-dimensional search space (line 8). The fitness function is then applied to assess the quality of the solution (line 9). The specific form of the fitness function depends on the optimization objective being tackled.

5. The crows' memory is updated based on the new solution (line 10):

$$m_{i,\text{iter}+1} = \begin{cases} x_{i,\text{iter}+1} & \text{if } f(x_{i,\text{iter}+1}) \leq f(m_{i,\text{iter}}) \\ m_{i,\text{iter}} & \text{otherwise} \end{cases} \tag{4-2}$$

where $f(x_{i,\text{iter}+1})$ and $f(m_{i,\text{iter}})$ represent the fitness values of, respectively, the new candidate position and the best food hiding spot for crow x_i.

It is important to note that exploration in the algorithm takes place when a crow chooses not to follow any other crow and instead moves toward a randomly selected position. In contrast, exploitation occurs when a crow follows another crow in an attempt to reach its food hiding place. The intensity of this exploitation is controlled by the flight length parameter, fl, as illustrated in Figure 4-3. When $fl < 1$, the crow targets a position somewhere between its current location (indicated by the blue dot) and the other crow's food hiding spot (orange dot). Conversely, when $fl > 1$, the crow may overshoot the hiding location and land beyond it. The exact landing point along this path, represented by the green arrow, is determined by the random number r_i, as described in Equation 4-1. This mechanism introduces a degree of uncertainty, intentionally blurring the boundary between exploration and exploitation. Such ambiguity prevents the algorithm from converging prematurely to local optima, which would occur if a crow always landed precisely at another crow's food hiding place.

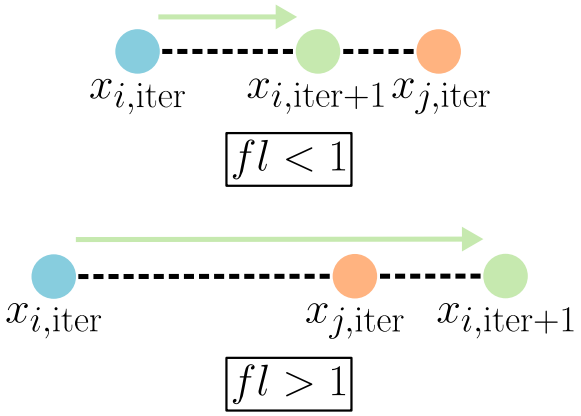


Figure 4-3: Exploitation process of the crows.

4.3.4 Particularization of CSA to the Scheduling Problem

The original CSA algorithm has undergone minor changes to be adapted to the target scheduling problem of this Thesis.

The optimization variables in this problem correspond to the tasks selected for hardware acceleration. Each variable can take values representing the number of parallel instances (i.e., accelerators) to deploy for a specific task, with values ranging from zero up to the total number of reconfigurable slots in the FPGA. Figure 4-4 illustrates this solution space for a simplified example where only two tasks can be scheduled and at most three slots are available. In such a case, the crows navigate a 2D solution space where each coordinate represents a valid 2-task combination using tasks A and B. The tasks assigned zero resources are considered in the next decision.

Unlike the original formulation of the CSA, the search space in this context is discrete, since task replication levels are restricted to integer values. To accommodate this, the proposed version of CSA incorporates a boundary-checking step (line 8

in Algorithm 4-1) that rounds each crow's next position to the nearest integer. Additionally, the combined total of all task replicas (i.e., the sum of all optimization variables) must not exceed the number of available FPGA slots. This constraint is enforced during boundary checking by decrementing the task with the largest allocation until the slot limit is satisfied (ensuring that those tasks assigned to only one accelerator are not completely removed).

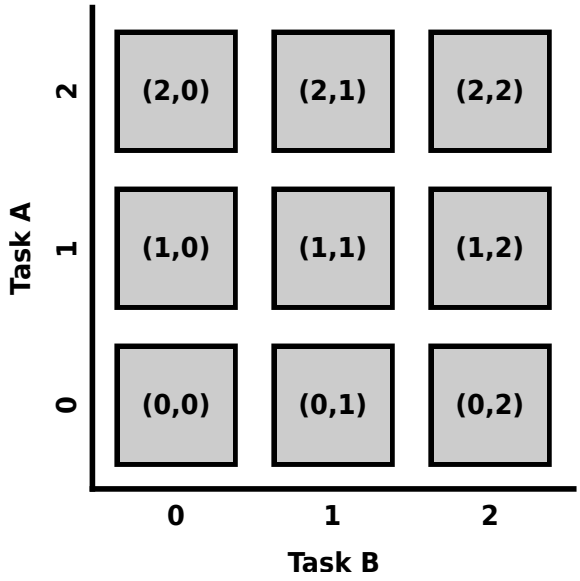


Figure 4-4: Dummy scheduling solution space (2 tasks, 3 slots).

With these changes on the CSA algorithm in place, the workload optimization process on the FPGA, as depicted in Figure 4-5, is then carried out as follows (note that the example kicks off at a random instant where certain tasks are already running, others are waiting in a queue, and some free slots are available):

1. Whenever a new task arrives or any reconfigurable FPGA slot becomes available in the FPGA fabric, the scheduler examines the queue of pending tasks to determine which ones should be deployed. This initiates the scheduling process.

2. The tasks awaiting execution act as optimization variables in a d-dimensional search space, where d corresponds to the number of tasks in the queue.

3. The CSA is executed for a predefined number of iterations using a set of crows, each searching a portion of the solution space. Each crow's position is evaluated using a fitness function (which will be detailed in the next subsection).

4. The task configuration associated with the position of the crow achieving the best fitness score is selected for deployment on the FPGA.

5. This routine is repeated indefinitely.

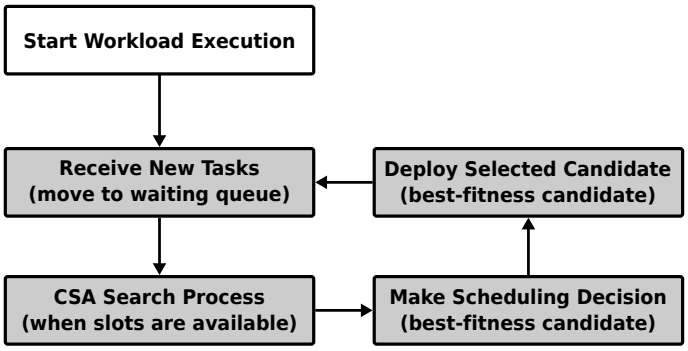


Figure 4-5: Flowchart of the proposed workload optimization strategy.

4.3.5 Fitness Function

The fitness function assesses the quality of each solution identified by the CSA and is central to the overall performance of the scheduler. In this Thesis, a multi-objective fitness function is introduced to optimize the two primary metrics frequently addressed in the literature: workload makespan (and consequently, energy efficiency, since it correlates with execution time for FPGAs) and fairness in resource allocation (i.e., equitable distribution of resources among tasks). In this formulation, lower fitness scores correspond to better scheduling decisions.

Workload Makespan Optimization: as previously discussed, the proposed scheduler makes use of the run-time workload characterization described in Chapter 3 to estimate the execution time for different combinations of tasks. Specifically, when a crow's position (representing an FPGA configuration) is evaluated, the predictive models are employed to estimate: (i) the execution time of each task in the waiting queue if scheduled at that specific moment, and (ii) the execution time the same task would require if executed in isolation. The relative difference between these two estimations indicates the predicted performance degradation caused by task interaction for each individual task in the current scheduling scenario. This interaction effect is computed for every task in the queue, contextualized for the configuration defined by the crow's position, since interaction levels differ across tasks and setups. When multiple tasks are proposed for execution, the average of these interaction values is calculated, resulting in an aggregated conflict-aware scheduling term.

To illustrate this aggregation, suppose that three tasks `A`, `B`, and `C` are scheduled together in a given configuration. Their predicted execution times when running alone are 100 ms, 120 ms, and 80 ms, respectively. When co-scheduled, the predicted times increase to 130 ms, 150 ms, and 100 ms. The relative degradations are therefore:

$$\texttt{A}: \ \frac{130-100}{100} = 0.30, \quad \texttt{B}: \ \frac{150-120}{120} = 0.25, \quad \texttt{C}: \ \frac{100-80}{80} = 0.25.$$

According to the proposal in this Thesis, the aggregated conflict-aware term used to assess this configuration is the average:

$$\frac{0.30 + 0.25 + 0.25}{3} = 0.2667.$$

This value is then incorporated into the objective function, promoting configurations with lower predicted contention.

Nonetheless, a trivial solution involving the scheduling of no tasks would yield zero conflict. To discourage the scheduler from proposing such inactive configurations as candidate solutions, an additional reward component is introduced to favor proposals with greater accelerator utilization, which is computed as the ratio between used and total reconfigurable slots for a given candidate. Together, these two elements form a comprehensive conflict term, which is detailed in Equation 4-3. Higher values for this term indicate poorer scheduling outcomes.

$$\text{Conflict} = \underbrace{\frac{1}{N}\sum_{i=1}^{N}\left(\frac{T_i^{\text{shared}} - T_i^{\text{alone}}}{T_i^{\text{alone}}}\right)}_{\text{Average Interaction Impact}} - \underbrace{\frac{Slots_{used}}{Slots_{total}}}_{\text{Activity Reward}} \tag{4-3}$$

Fair Resource Allocation: the second component of the multi-objective fitness function assesses how evenly the available resources are distributed among tasks. For each candidate solution (i.e., each crow position), the standard deviation of the optimization variable vector, representing the replicas assigned to each task, is calculated as defined in Equation 4-4. A larger standard deviation reflects an uneven allocation, where some tasks are favored with more replicas while others receive fewer or none. This imbalance indicates a lack of fairness in resource usage. Similar to the conflict term, higher values in this metric correspond to poorer scheduling outcomes.

$$\text{Fairness} = \text{std}\left(\{Slots_{T_1}, Slots_{T_2}, \ldots, Slots_{Tn}\}\right) \tag{4-4}$$

the overall fitness score is computed as a weighted linear combination of the normalized aggregated conflict and fairness terms, governed by a user-defined parameter α, as described in Equation 4-5. Normalizing both components ensures that their combination is adaptable to different optimization priorities. By adjusting α, the user can steer the behavior of the scheduler: higher values place greater emphasis on minimizing task interaction, whereas lower values give preference to a fairer distribution of the FPGA resources.

$$\text{Fitness} = \alpha \times \text{Conflict} + (1 - \alpha) \times \text{Fairness} \tag{4-5}$$

Original contribution 4-2 *A task scheduling policy that uses run-time kernel interaction estimations to guide its scheduling decisions in reconfigurable multi-accelerator systems.*

4.4 Validation

This section covers the evaluation of the proposed conflict-aware scheduler. First, the experimental setup used for this validation campaign is described. Next, Pareto and sensitivity analyses of the scheduler parameters are conducted in a simulation environment to select the most appropriate values, also illustrating their impact on the scheduler behavior. The section concludes by analyzing the results obtained with the proposed scheduler in real hardware, followed by a brief assessment of its portability.

4.4.1 Experimental Setup

The experimental setup for this validation campaign is essentially the same as the one presented in Section 3.7.2. The testing platform is the Zynq UltraScale+ ZCU102,[4] containing 8 reconfigurable slots to be used for scheduling hardware acceleration requests. Regarding the workload to be optimized, the MachSuite benchmarks are used as the hardware acceleration tasks. Note that to ensure the workload presents a scenario comparable to a cloud-edge continuum environment, the dynamic workload has been generated with the synthetic workload generator described in Chapter 3. Specifically, the workload includes 60,000 hardware acceleration requests, which incorporate additional kernel functions gradually over time. Initially, the first 20,000 requests feature only four kernels, followed by an additional 20,000 that include four more kernels, for a total of 8. The last 20,000 requests include all 11 available kernels.

4.4.2 Selection of the Scheduler Parameters

The scheduler outlined in Section 4.3 depends on a set of tunable parameters that affect both the efficiency of the optimization process and the quality of the scheduling decisions. Among these parameters are the population size (i.e., number of crows), the maximum number of optimization iterations, and the core CSA parameters: awareness probability (AP) and flight length (fl), which govern the balance between exploration and exploitation. In addition, the trade-off coefficient α, responsible for weighting the relative importance of task conflict versus fair use of resources, plays a critical role. Lastly, the number of tasks (from the waiting queue) to be considered as scheduling candidates must be specified, which primarily affects the scheduler's convergence time, as it increases the solution space. Table 4-2 lists each of these tunable parameters.

To calibrate these parameters, a simulation environment has been created to emulate workload execution using the predictive models introduced in Chapter 3. Since the objective here is to determine effective algorithm configurations rather than to carry out live scheduling, this evaluation employs a pre-trained model. The simulator reconstructs a detailed timeline of task execution, accounting for overlaps

[4] XCZU9EG-FFVB1156-2-I device.

Table 4-2: Tunable parameters in the scheduler

Parameter	Description
Number of Crows	Size of the CSA population controlling search diversity
Number of Iterations	Maximum number of optimization iterations
Awareness Probability (AP)	Probability that a crow detects being followed (exploration)
Flight Length (fl)	Distance a crow moves toward another crow (exploitation)
Trade-off Coefficient (α)	Weight factor between conflict minimization and fairness
Number of Candidate Tasks	Tasks selected for scheduling (simultaneously)

and task interactions. When a task starts or finishes, the estimated end times of other concurrent tasks are updated using the predictive models, thereby capturing the effects of task interactions in a realistic manner. Figure 4-6 presents a flow diagram of how the simulation operates. This approach allows for a fair assessment of alternative scheduling strategies in a controlled, yet practical, setting.

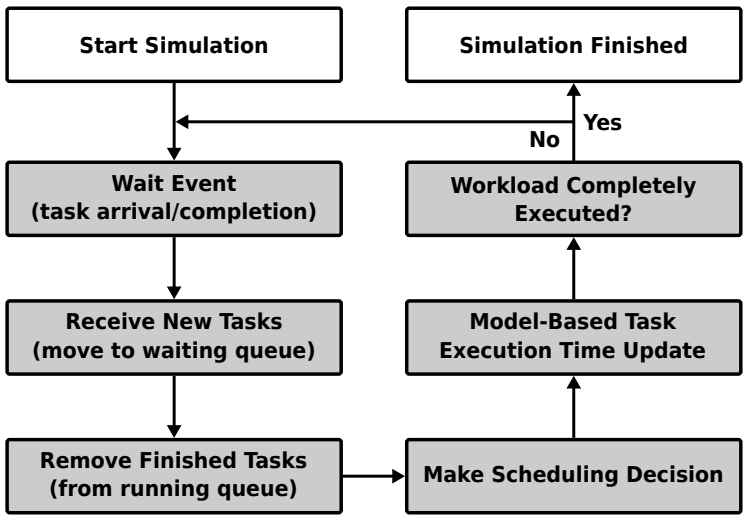


Figure 4-6: Flowchart of the workload execution simulator.

With this simulation framework, a grid search has been conducted to find combinations of the scheduling parameters introduced in Section 4.3 that lead to consistent scheduling performance. The grid search process runs multiple scheduling trials on the described synthetic workload, systematically varying the configuration parameters of the scheduler in each run, obtaining different optimization results. The goal is to find parameter combinations that offer the best balance between scheduling quality and optimization efficiency. Specifically, the analysis targets configurations that minimize workload makespan while reducing optimization time. Guided by prior empirical observations, the search space is limited to the values shown in Table 4-3. Each grid candidate (i.e., a scheduler configuration) is evaluated using three metrics: scheduling time (overhead), workload makespan (first optimization objective), and average task wait time (fairness, second optimization objective).

Table 4-3: Parameter search space for sensitivity analysis of the conflict-aware scheduler

Scheduler Parameters	**Evaluated Values**
Trade-off Coefficient (α)	[0.1, 0.3, 0.5, 0.7, 0.9]
Number of Candidate Tasks	[1, 2, 3, 4]
CSA Control Parameters	**Evaluated Values**
Number of Crows	[1, 2, 3, 4, 5, 6]
Number of Iterations	[2, 4, 6, 8, 10]
CSA Intrinsic Parameters	**Evaluated Values**
Awareness Probability (AP)	[0.2, 0.4, 0.6, 0.8]
Flight Length (fl)	[0.5, 1.5, 2.5, 3.5, 4.5, 5.5]

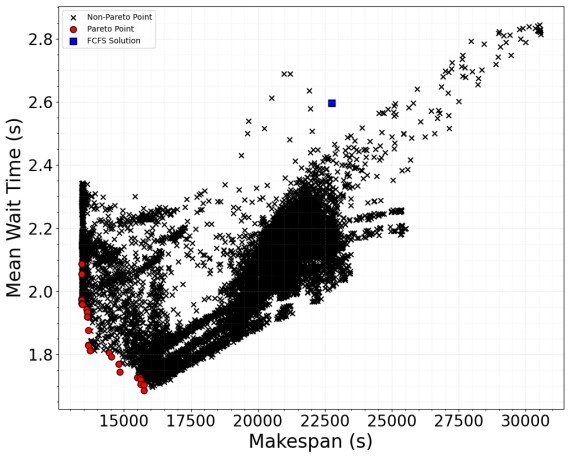
4.4.3 Pareto Front Analysis

The Pareto plots in Figure 4-7 show the results of the grid search. Each point represents a specific combination of the evaluated scheduling parameters. The optimal trade-offs, corresponding to the Pareto fronts, are highlighted as red dots in each graph. Moreover, since no other alternatives have been found as part of the review of the state of the art for a comparable problem, a FCFS scheduling approach has been also simulated and used as the baseline solution, where tasks are executed in order of arrival selecting a random number of replicas per task from a uniform distribution between 1 and the maximum number of available execution slots (i.e., 8 in this case). This reference example is represented as a blue square.

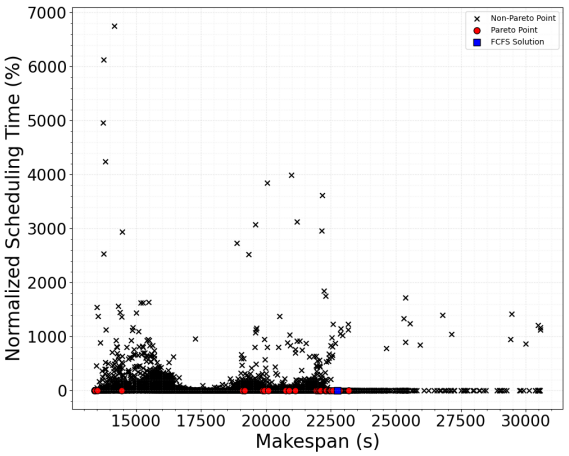
Figure 4-7a illustrates the trade-off between the average task wait time and the workload makespan. The results show that varying the parameters leads to different compromises between these two metrics. However, no single configuration clearly dominates. As you move towards the optimal region (i.e., the lower-left corner, where both makespan and wait time are minimized), further reduction in makespan tends to increase wait time, and vice versa, especially around the Pareto front. This behavior is as expected, since the fitness function favors conflict minimization, which may delay some tasks in favor of more efficient scheduling opportunities. This can lead to longer wait times, but overall, better workload makespans.

In contrast, Figure 4-7b shows the trade-off between scheduling time (i.e., optimization overhead), normalized to the workload makespan, and the makespan itself. Note that the makespan shown in this figure does not include the scheduling time, allowing for a decoupled comparison. The actual execution time would be the sum of both values.

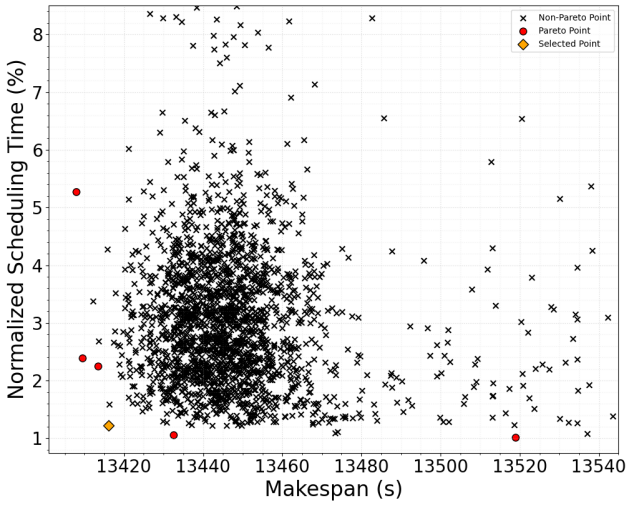
A zoomed-in view is provided in Figure 4-7c. This zoom focuses on the most promising configurations (i.e., those located towards the lower-left corner, characterized by low makespan and low overhead). Unlike the wait time vs. makespan trade-off, a clear optimal configuration emerges here, marked by a yellow diamond.



(a) Trade-off – Wait time vs. makespan.



(b) Trade-off – scheduling time vs. makespan.



(c) Zoom-in trade-off – scheduling time vs. makespan.

Figure 4-7: Sensitivity analysis of the scheduler parameters.

This point offers the best balance between scheduling overhead and makespan. The fair use of resources could be selected as the optimization goal as it is allowed by the proposed fitness function, but for this Thesis, makespan optimization (and indirectly energy efficiency) is prioritized.[5] Hence, this configuration (the one highlighted with the yellow diamond) has been selected as the operating point of the scheduler for the subsequent hardware validation tests, described in the next section. The exact parameter values for this configuration are listed in Table 4-4.

[5]The cloud-edge continuum scenario under evaluation in this Thesis does not assume any real-time constraints in the workload, so it makes sense to prioritize both makespan and energy.

Table 4-4: Conflict-aware scheduler parameters selection

Scheduler Parameters	**Selected Values**
Trade-off Coefficient (α)	0.9
Number of Candidate Tasks	3
CSA Control Parameters	**Selected Values**
Number of Crows	3
Number of Iterations	4
CSA Intrinsic Parameters	**Selected Values**
Awareness Probability (AP)	0.8
Flight Length (fl)	5.5

4.4.4 Sensitivity Analysis

Besides what has been said in the previous section, if one wishes to prioritize the fair use of resources (i.e., reducing task wait times), a different trade-off can be achieved by adjusting the scheduler parameters. There is a wide range of trade-off solutions depending on user needs, most of which deliver better results than the baseline FCFS solution. In fact, an FCFS approach yields, on average, significantly worse performance than most of the configurations from the grid search (see blue square in Figure 4-7a). Therefore, a sensitivity analysis has been performed to analyze the impact of the configuration parameters on the optimization results. The following sections will illustrate how specific parameter choices influence the solutions and how workload optimization responds to changes in each parameter. The sensitivity analysis will be structured as follows:

1. The evaluated parameters are grouped into three categories: (i) general scheduler parameters (namely, α and the number of candidate tasks), (ii) CSA control parameters (i.e., number of crows and number of iterations), and (iii) CSA intrinsic parameters (i.e., awareness probability AP and flight length fl).

2. For each parameter, a set of plots, analogous to those in Figure 4-7, will be shown to illustrate how varying that parameter affects the outcomes. Additional plots will be included where necessary to provide further insights. These results will be discussed in depth. Although the primary objective of this Thesis is to minimize workload makespan, and scheduling overhead, the discussion will also reflect on how the approach could be used to enforce the fair use of resources.

3. After analyzing each parameter, the best-performing value will be selected and fixed before moving on to evaluate the next one. To ensure consistency and interpretability in the analysis, parameters are examined in a specific order, from general to specific. This means starting with high-level scheduler settings,

such as the trade-off coefficient (α) and the number of task candidates, and then progressing to more algorithm-specific parameters, like the awareness probability (AP) and flight length (fl) of the CSA. This ordering is essential because the effect of low-level parameters often depends on the broader scheduling strategy defined by the higher-level ones. For example, the optimal value of AP may vary significantly depending on the value of α. If AP were to be analyzed before fixing α, some conclusions might be drawn that no longer hold valid once α is changed. In other words, evaluating more specific parameters without first fixing the general context would make the results harder to interpret and potentially misleading. By progressively refining the configuration, starting from the most influential parameters, it is ensured that each evaluation has a solid foundation, enabling reliable comparison and selection at each stage.

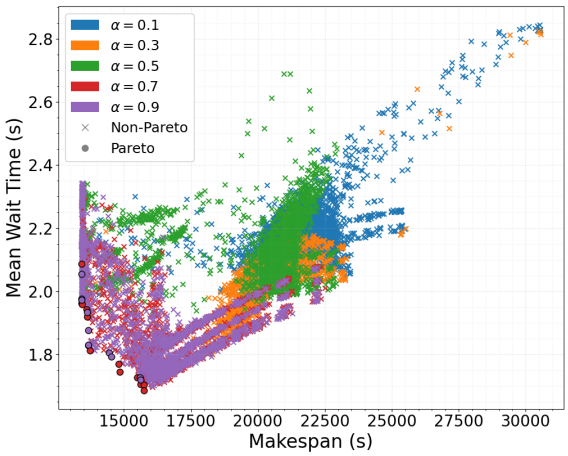
Scheduler Parameters

The first parameter under evaluation is α, which biases the fitness function towards either minimizing task conflicts or promoting fair resource usage. Figure 4-8 illustrates the trade-offs associated with different α values, each shown in a distinct color.

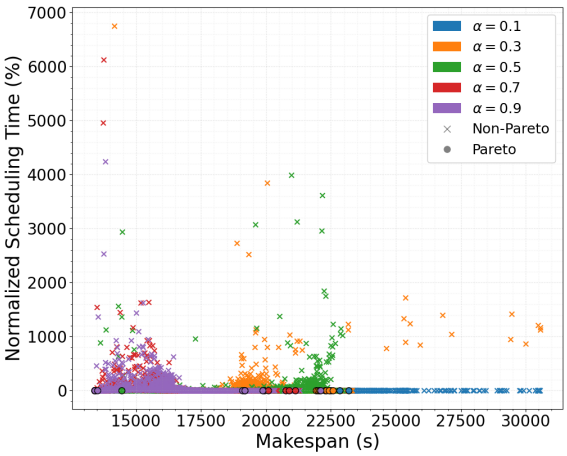
Both Figure 4-8a and Figure 4-8b show that smaller values of α (e.g., $\alpha = 0.1$ or 0.3), which impose fair resource utilization over conflict minimization, result in worse overall performance, with higher wait times and longer makespans. In contrast, larger values of α (especially $\alpha \geq 0.7$), which assign more weight to conflict minimization, produce significantly better results for both metrics. This trend demonstrates that attempting to maintain fairness by ensuring all tasks get their share of the reconfigurable logic ultimately degrades system performance due to the associated increase in task interactions and resource contention.

Within the region of higher α values, the differences between configurations become more subtle. Figure 4-8a shows that configurations minimizing wait time tend to occur at $\alpha = 0.7$, while those minimizing makespan tend to occur at $\alpha = 0.9$. This is confirmed in Figure 4-8c, which reveals that the lowest makespans (with minimal scheduling overhead) correspond to $\alpha = 0.9$. Therefore, since the objective with the highest priority in this Thesis is makespan reduction, the value $\alpha = 0.9$ is selected for all subsequent analyses. For other user needs, such as fair use of resources, lower values of α could be selected. Note that the yellow diamond in the figure, indicating the overall selected configuration for hardware evaluation, also corresponds to this value, reinforcing the consistency of this choice.

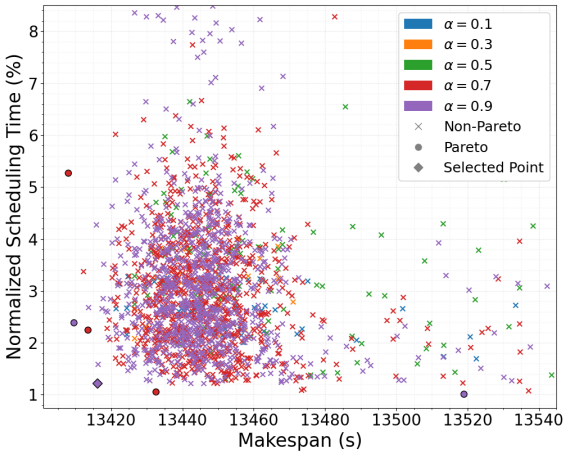
The next parameter analyzed is the number of candidate tasks considered for scheduling. That is, how many tasks in the waiting queue are evaluated as potential candidates during each scheduling decision. The results are shown in Figure 4-9, where the impact of this parameter is evident across all metrics: wait time, makespan, and scheduling overhead.



(a) Trade-off – Wait time vs. makespan.



(b) Trade-off – scheduling time vs. makespan.

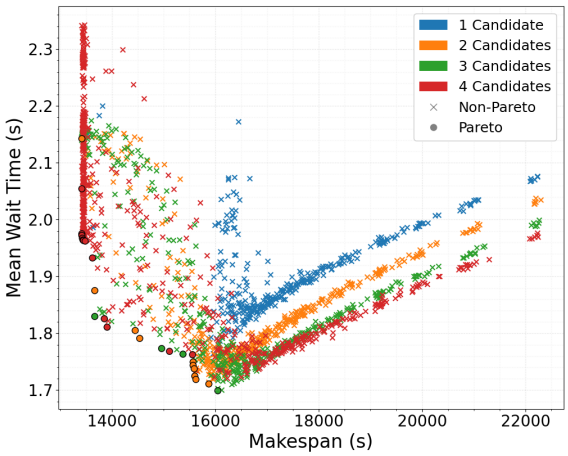


(c) Zoom-in trade-off – scheduling time vs. makespan.

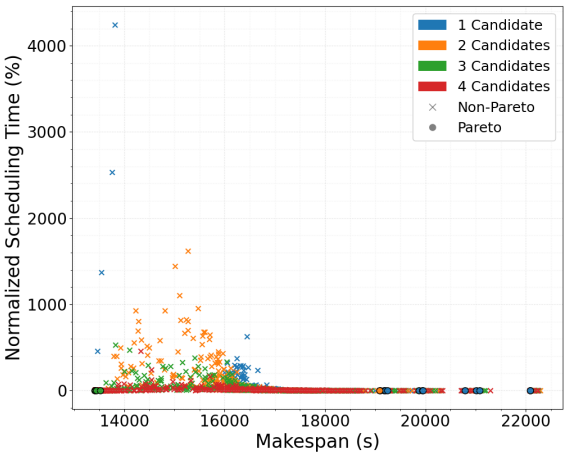
Figure 4-8: Sensitivity analysis of the scheduler parameters – Impact of α.

In terms of wait time, Figure 4-9a reveals a striped pattern that indicates a strong correlation between the number of candidates and wait time: as more tasks are considered, the more their wait time tends to decrease. This makes sense, as evaluating more options allows the scheduler to exploit task-level parallelism better, thus reducing the time tasks spend waiting. However, once a certain point is reached (around a makespan of 16,000 seconds), further reductions in makespan are accompanied by increased wait times. This occurs because minimizing makespan often involves prioritizing low-conflict tasks, pushing others to later time slots. This behavior could be mitigated, if needed, by introducing additional mechanisms to ensure fairness in the fitness function, such as aging.

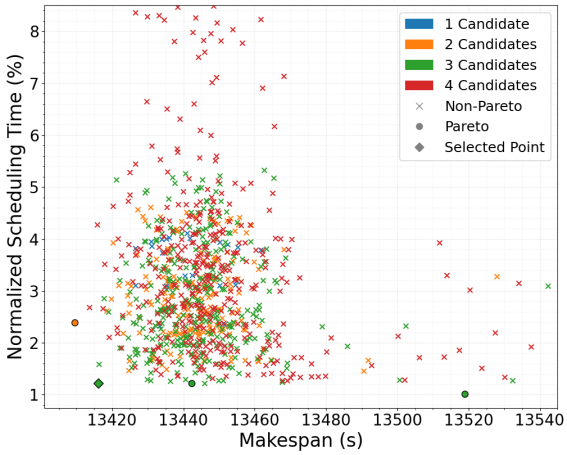
On the makespan side, the best results (i.e., <16,000 s) are obtained when considering three to four candidate tasks. Figure 4-9c confirms that these configurations offer the best makespan-to-overhead balance. However, increasing the number of candidates also increases optimization overhead. The reason is that



(a) Trade-off – Wait time vs. makespan.



(b) Trade-off – scheduling time vs. makespan.



(c) Zoom-in trade-off – scheduling time vs. makespan.

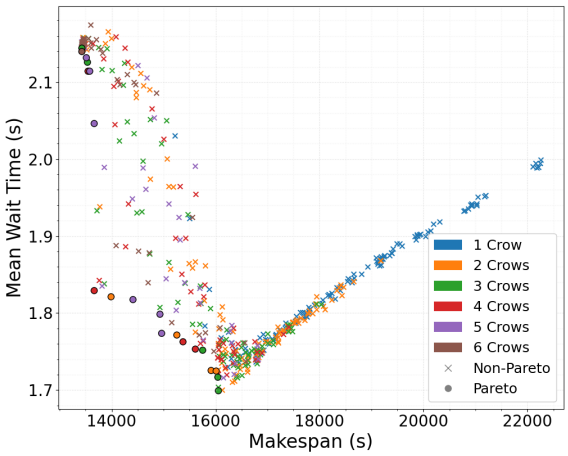
Figure 4-9: Sensitivity analysis of the scheduler parameters – Impact of candidates number.

a greater number of candidate tasks to be evaluated translates to a search space with more dimensions, and therefore, a longer convergence time (i.e., scheduling overhead). Consequently, a moderate value of three candidates is selected, which still aligns with the configuration marked by the diamond in the figure.

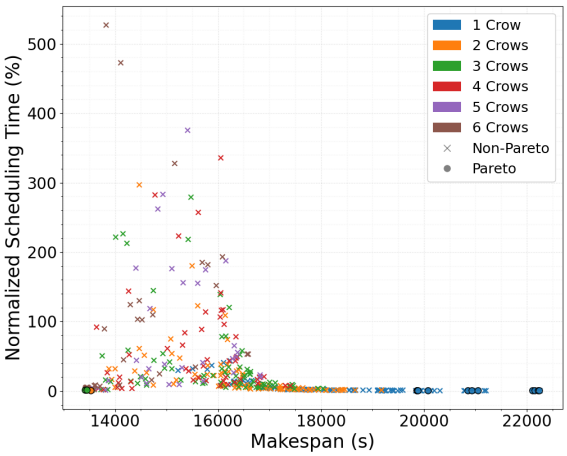
CSA Control Parameters

This section evaluates the parameters that govern how the CSA operates: the number of crows and the number of optimization iterations.

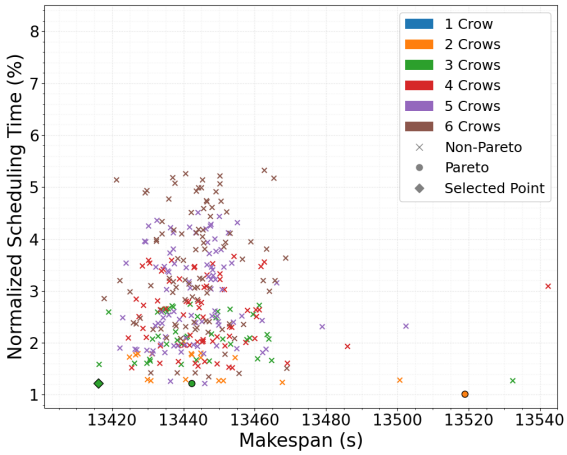
With respect to the number of crows, Figure 4-10 presents the trade-offs observed when this parameter is varied. A clear pattern emerges in both Figure 4-10a and Figure 4-10b: increasing the number of crows generally leads to lower makespans. This is because a larger crow population explores a wider part of the solution space, increasing the likelihood of discovering better scheduling configurations.



(a) Trade-off – Wait time vs. makespan.



(b) Trade-off – scheduling time vs. makespan.

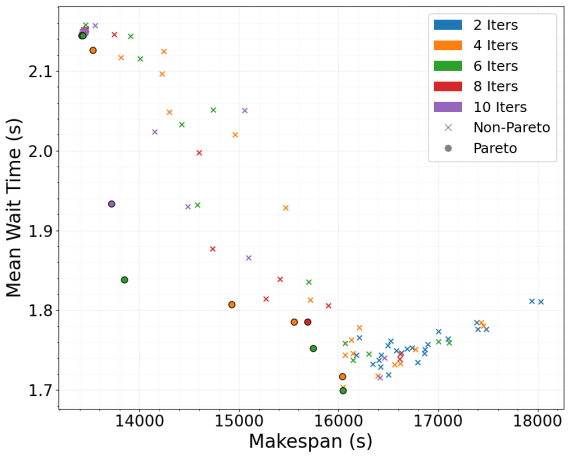


(c) Zoom-in trade-off – scheduling time vs. makespan.

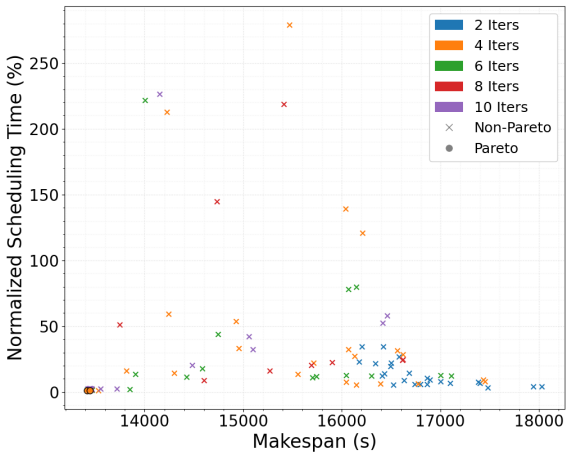
Figure 4-10: Sensitivity analysis of the scheduler parameters – Impact of crow population.

As previously discussed, a reduction in makespan typically also results in shorter wait times, up to a certain point beyond which wait times may rise again due to the mentioned trade-off dynamics. Focusing on the makespan versus scheduling overhead trade-off shown in Figure 4-10c, a similar effect is observed: while more crows improve solution quality, excessive crow counts significantly increase optimization time, leading to higher scheduling overhead. Therefore, a balanced choice, specifically, using three crows, emerges as the most efficient configuration. This setting represents the best Pareto point in the plot and is, in fact, the value selected for the configuration to be validated in hardware.

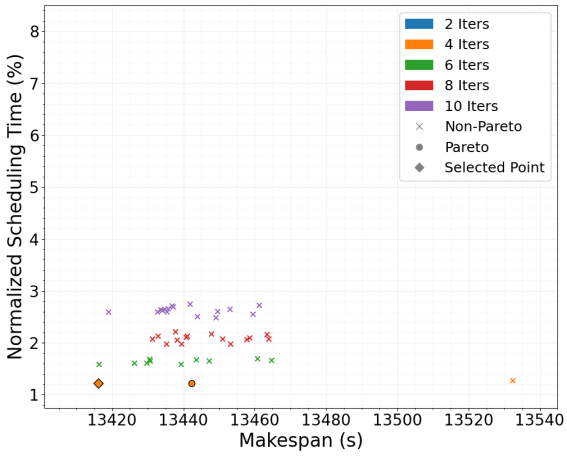
Regarding the number of iterations, the trade-offs are illustrated in Figure 4-11. As previously observed, too few iterations (e.g., two) yield poor results, especially in terms of makespan, due to limited exploration of the solution space. On the other hand, increasing the number of iterations improves solution quality but also increases overhead. As shown in Figure 4-11c, there is a layered structure based on the number



(a) Trade-off – Wait time vs. makespan.



(b) Trade-off – scheduling time vs. makespan.



(c) Zoom-in trade-off – scheduling time vs. makespan.

Figure 4-11: Sensitivity analysis of the scheduler parameters – Impact of iterations.

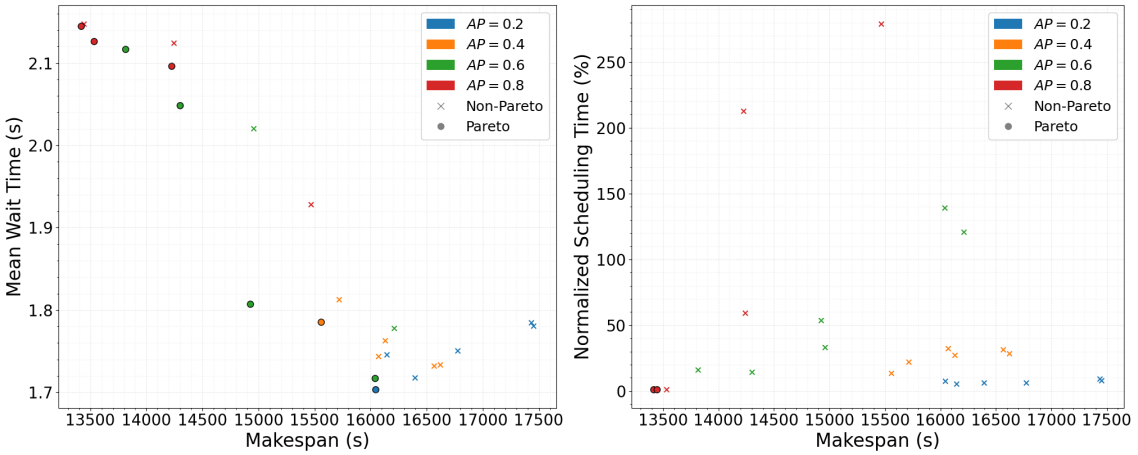
of iterations. Most render similar makespan results, but four iterations provide the best compromise between quality and efficiency (i.e., less overhead). This value is therefore adopted for the remainder of the analysis and is again consistent with the configuration chosen for hardware evaluation.

CSA Intrinsic Parameters

Finally, the intrinsic parameters of the CSA algorithm, awareness probability (AP) and flight length (fl), are analyzed. These parameters directly influence how the crows behave during the optimization process.

Starting with AP, which defines how often a crow decides to explore a new position (with higher values promoting exploration), Figure 4-12 presents the corresponding trade-offs. As seen in Figure 4-12b, higher AP values (e.g., 0.8 or 0.9) lead to better makespan outcomes. This is because increased exploration helps the algorithm avoid

getting stuck in local optima. This effect is significant in contexts with a low number of iterations, as in this Thesis, where over-exploitation can cause limited exploration of the solution space and local-optima solutions. However, higher AP values also lead to increased wait times (as shown in Figure 4-12a) due to the trade-off behavior found around the 16,000 s threshold.



(a) Trade-off – Wait time vs. makespan.

(b) Trade-off – scheduling time vs. makespan.

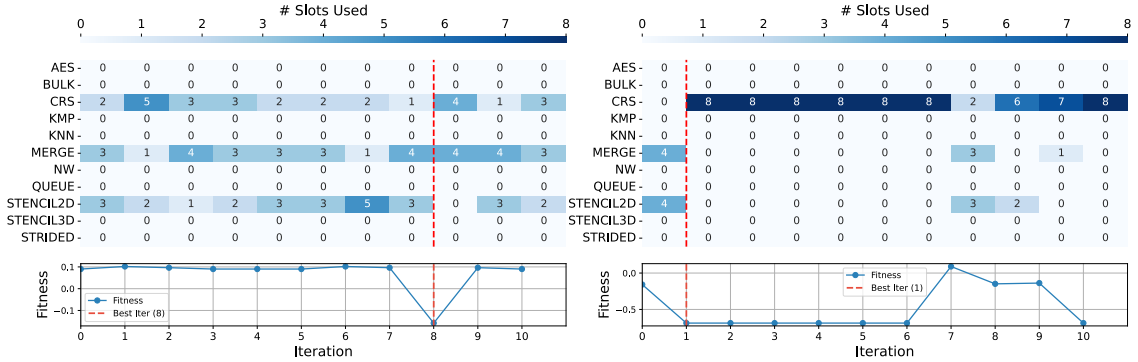
Figure 4-12: Sensitivity analysis of the scheduler parameters – Impact of AP.

To illustrate better how the variation of the AP parameter affects the behavior of the crows, Figure 4-13 shows the trajectory of a single crow throughout the optimization for a case involving three queued tasks (CRS, MERGE, and STENCIL2D) scheduled on an FPGA with eight free slots. The upper subplot depicts the assignment of accelerators to each task over time (i.e., optimization iterations), while the lower one displays how the fitness value evolves throughout the process. Note that this serves only as a representative example where, since just three tasks are involved, the rest feature zero accelerators. These results demonstrate that higher AP values (Figure 4-13a) increase exploratory behavior, making crows more evasive to test more diverse alternatives, whereas lower AP settings (Figure 4-13b) favor faster convergence by allowing crows to be easily followed. Please note that the red vertical line indicates which solution candidate is selected for scheduling.

Given the limited number of iterations, the best value of AP to achieve a low makespan is 0.8, as confirmed by Figure 4-12b. This value has therefore been adopted in the final configuration.

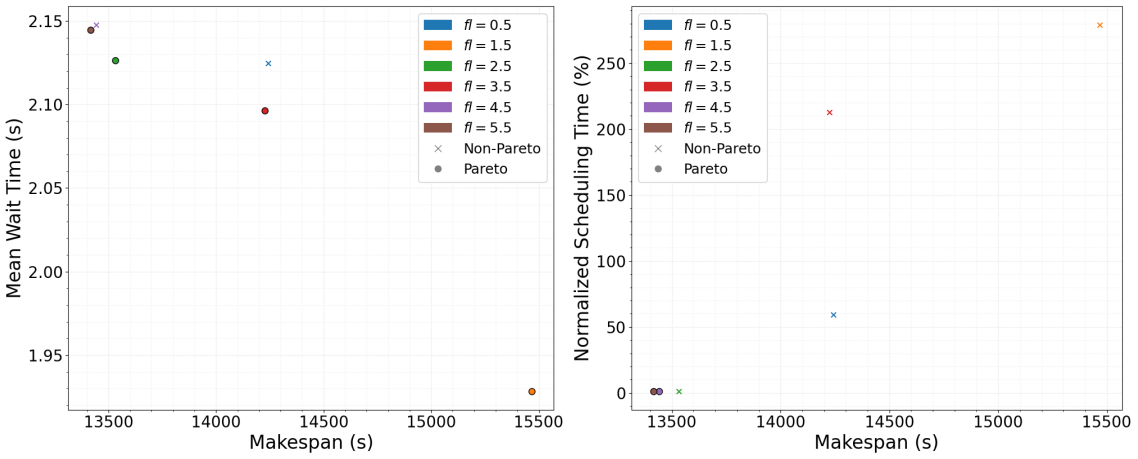
Lastly, the impact of flight length fl, which determines how far a crow can move when following another, is shown in Figure 4-14. Similar to AP, higher fl values provide better makespan performance by enabling broader exploration of the solution space. Again, this behavior is advantageous when the optimization time (in terms of iterations) is limited, as is the case here.

To better understand the influence of the fl parameter on the solution space searching process, an example where AP is fixed to 0.1 to encourage stronger



(a) $AP = 0.9$ **(b)** $AP = 0.1$

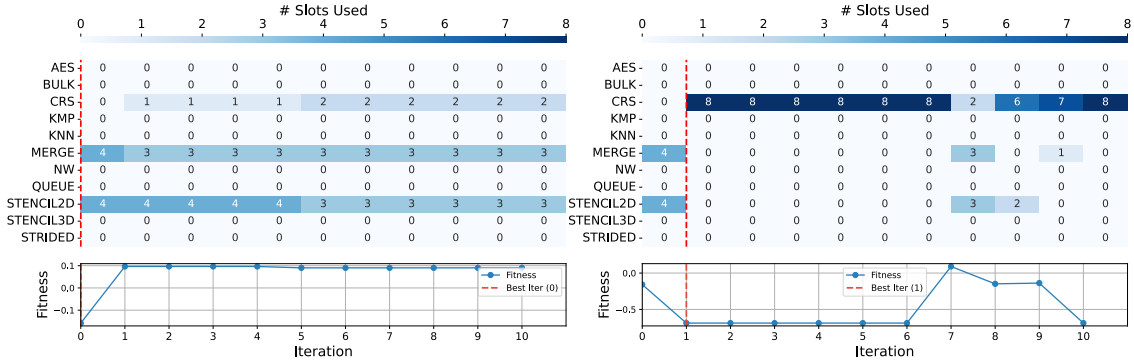
Figure 4-13: Impact of the awareness probability (AP) on the behavior of the CSA search process. The red vertical line represents the solution candidate selected for scheduling.



(a) Trade-off – Wait time vs. makespan. **(b)** Trade-off – scheduling time vs. makespan.

Figure 4-14: Sensitivity analysis of the scheduler parameters – Impact of fl.

following behavior among crows is presented. This configuration ensures that position updates are primarily driven by the memory of other individuals, rather than random exploration, which is where fl plays a key role. Figure 4-15 illustrates the trajectory of a single crow throughout the optimization process under different fl settings, in a scenario with three queued tasks (CRS, MERGE, and STENCIL2D) targeting an FPGA with eight available slots. With a low fl value (Figure 4-15a), the crow exhibits only one-by-one movements, often falling short when reaching distant positions and instead stopping halfway. In contrast, higher fl values (Figure 4-15b) allow the crow to perform longer jumps, enabling it to explore more distant and potentially better regions of the search space. See, for instance, how in the first iteration of Figure 4-15b the crow jumps from a position where CRS has zero replicas, directly to eight replicas, which happen to give the best fitness so far. This highlights the critical role of fl in balancing local exploitation and global exploration.



(a) $fl = 0.5$ **(b)** $fl = 5.5$

Figure 4-15: Impact of the flight length (fl) on the behavior of the CSA search process. The red vertical line represents the solution candidate selected for scheduling.

In optimization scenarios like the one presented in this Thesis, where the limited optimization time requires a reduced number of optimization iterations, it is fundamental that the search algorithm can move directly to remote positions to enable a faster exploration of distinct parts of the solution space, even though that might result in less local exploitation. In other words, the best value of fl, which is confirmed by rendering the best makespan in Figure 4-14b, is 5.5. This has consequently been adopted in the final configuration for the remainder of the Thesis.

Ending Remarks

The results obtained with the sensitivity analysis, guided by the goal of minimizing workload makespan rather than fair use of resources, have concluded that the most appropriate values for the scheduler parameters match those derived from the Pareto analysis and presented in Table 4-4. Moreover, the sensitivity analysis has provided valuable insights into how each setting impacts the overall system performance in terms of workload makespan, average task wait time, and scheduling overhead:

- **Trade-off coefficient (α):** higher values of α (particularly $\alpha = 0.9$) lead to better performance by focusing more on conflict minimization. This results in lower makespans and wait times, especially when compared to lower α values that attempt to enforce fairness at the cost of higher contention.
- **Number of candidate tasks:** considering more candidate tasks initially improves the makespan and reduces wait time, but this comes at a cost in terms of scheduling overhead due to a larger search space. A moderate value (three candidates) offers the best trade-off.
- **Number of crows:** increasing the number of crows enhances solution quality by expanding exploration, but too many lead to significantly higher optimization times. The best balance is achieved with a mid-range value of three crows.

- **Number of iterations:** a small number of iterations (e.g., two) is insufficient for effective optimization. On the other hand, increasing the number of iterations improves makespan but also raises overhead. A value of four iterations renders the best balance.

- **Awareness probability (AP):** higher values (e.g., $AP = 0.8$) promote exploration and help avoid local optima, especially important when using few iterations. However, they slightly increase wait time.

- **Flight length (fl):** similar to AP, larger flight lengths enhance global exploration by allowing crows to reach distant positions, which improves makespan under low iteration constraints. Lower values limit movement to short jumps, slowing convergence. The best results are obtained with $fl = 5.5$.

4.4.5 Workload Optimization Results

This section presents the evaluation of the conflict-aware workload optimization strategy and compares its performance and energy efficiency with the baseline method used in previous chapters. Moreover, even though the primary focus of this Thesis is makespan and energy efficiency optimization, an evaluation of the impact of the α parameter in workload optimization is presented, to explore the makespan vs. fairness trade-offs available at run time for the user. The scheduler is set up using the parameters derived from the Pareto and sensitivity analyses presented in the previous sections (refer to Table 4-4), and it is embedded within the workload management framework introduced in Chapter 2. The experiments are conducted on the hardware platform specified in the experimental setup, executing the synthetic workload defined in that section. The behavior of the proposed approach follows the operation described during this chapter, utilizing the run-time workload characterization described in Chapter 3 to assist its scheduling decisions.

For this evaluation, the proposed methodology is compared against an FCFS reference, as state-of-the-art works do not cover run-time workload optimizations for scenarios like the one targeted in this Thesis. In this baseline, tasks are executed in their arrival order, and the number of replicas per task is randomly chosen from a uniform distribution between 1 and the number of available execution slots (i.e., 8 in this case).

Makespan Evaluation

Figure 4-16 provides a visual comparison of the workload makespan for both the proposed optimization method and the FCFS baseline, with all values normalized to the baseline. For the proposed workload optimization method, the reported makespan also includes the overheads associated with run-time workload characterization and optimization, which are not considered in the baseline case.

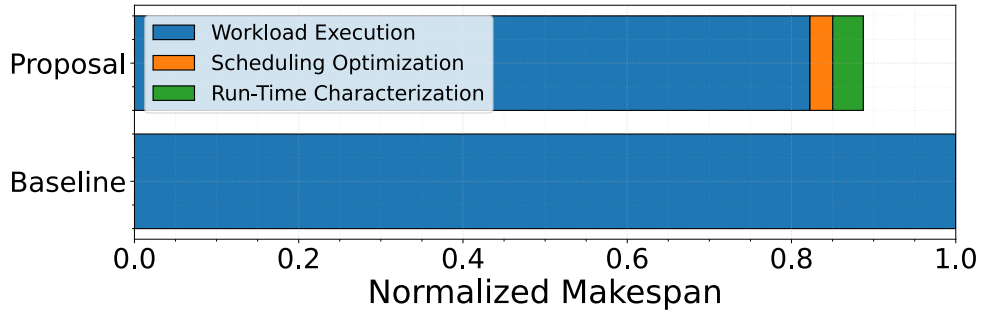


Figure 4-16: Normalized makespan comparison between the workload optimization methodology and the baseline approach.

The results indicate that in the ideal steady-state scenario, where the workload has already been characterized and remains unchanged (i.e., excluding the characterization overhead), the proposed workload optimization strategy achieves a workload execution equivalent to 84.94% of the baseline, corresponding to a 1.18× acceleration in workload execution. In case the workload does change and re-characterization is required (i.e., accounting for the characterization overhead), the results show that the optimization strategy achieves a workload makespan representing 88.59% of the baseline, corresponding to a 1.13× speedup. Whether the optimization achieved moves towards one side or the other will depend on the model updating frequency imposed by the nature of the workload.

As an example of this behavior, an experiment has been conducted in which the job size of each task is doubled, effectively doubling its execution time. This setup preserves the same number of scheduling decisions and the same amount of run-time characterization, but results in a longer workload execution. Consequently, the relative overhead of both characterization and scheduling is reduced, as shown in Figure 4-17. In this case, the proposed workload optimization strategy reduced the overall makespan to 83.14% of the baseline (a 16.86% reduction). These results show the optimization to be even better than the 84.94% mentioned in the ideal scenario, which happened due to the scheduling overhead being diluted over a longer workload execution. This scenario is representative of long-running workloads where new tasks appear infrequently, thus requiring fewer re-characterizations and allowing scheduling decisions to be made less frequently.

Nevertheless, returning to the initial example at Figure 4-16, even when accounting for the time spent on workload characterization and optimization, the proposed strategy still delivers an 11.41% reduction in workload makespan. Note that the run-time characterization included within this makespan can also be repurposed by a higher-level scheduler in the cloud-edge continuum to optimize global workload distribution, based on run-time node-level workload characterizations.

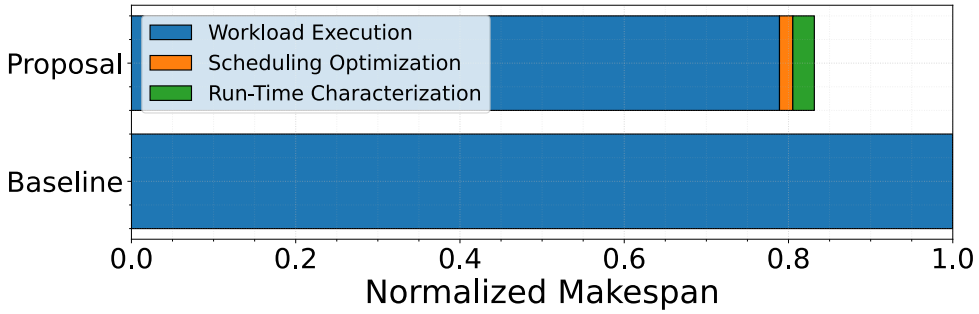


Figure 4-17: Normalized makespan comparison between the workload optimization methodology and the baseline approach when doubling the job size of each workload task.

Energy Evaluation

To evaluate energy consumption with each scheduling approach, it has been decided to take advantage of the monitoring framework described in Section 2.3.3. Periodic measurements of power consumption are taken during the execution of the workload using both scheduling approaches. The result is a series of monitoring windows that contain the power consumption of the system during that period of time. Specifically, a monitoring window has been measured every 500 ms, collecting data during a 100 ms. Then, the average power consumption for each monitoring window is calculated, and finally, a single average power consumption value is computed by averaging every monitoring window. Table 4-5 shows the power consumption (PS and PL power consumption) measured when running the workload for both the proposed workload optimization approach and the FCFS baseline alternative, using the mean and standard deviation.

Table 4-5: Comparison of the power consumption of the platform between the workload optimization approach and the baseline approach.

Scheduling Approach	PS		PL	
	Mean	**Std. Deviation**	**Mean**	**Std. Deviation**
Baseline	1.342 W	0.084 W	0.253 W	0.017 W
Proposal	1.254 W	0.042 W	0.256 W	0.181 W

It can be seen that the standard deviation is fairly low in all cases, indicating that there is not much variation in power consumption regardless of the specific decision taken in the schedule, which correlates with the analysis performed in Section 3.3. However, just for the PS, the mean power consumption is slightly lower when employing the proposed workload optimization methodology. The reason for this phenomenon is that when the task interaction is minimized, several contention sources are reduced (e.g., memory, CPU, DMA). This translates into the CPU not having to wait for those resources to be available, effectively lowering its duty cycle,

since the hardware execution remains the same. In contrast, the CPU operation is reduced. This lower CPU duty cycle results in lower average power consumption. The average power consumption of PL, in contrast, remains almost the same, as the hardware execution itself does not change.

Since power measurements were taken in periodic monitoring windows throughout the entire workload execution and provide stable values, the resulting mean power values can be used as an approximation for the overall average power consumption. To estimate energy consumption, the mean power value has been multiplied by the workload execution time shown in Figure 4-16, resulting in the total energy use by each scheduling approach. Finally, these energy values are normalized to the baseline approach and plotted for the PS and PL in Figure 4-18 and Figure 4-19, respectively.

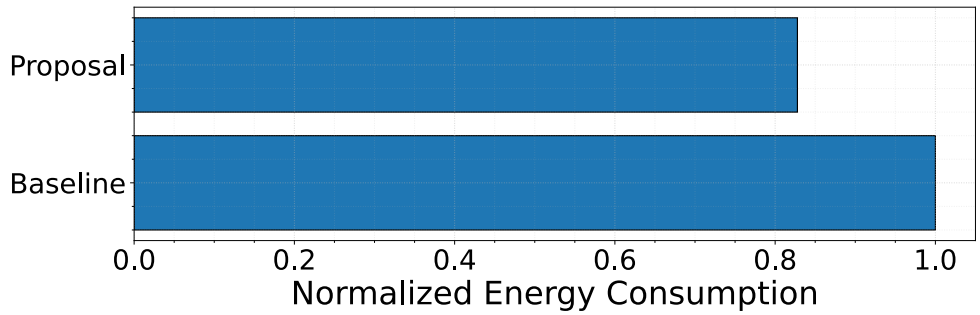


Figure 4-18: Comparison of the normalized PS energy consumed during workload execution between the workload optimization methodology and the baseline approach.

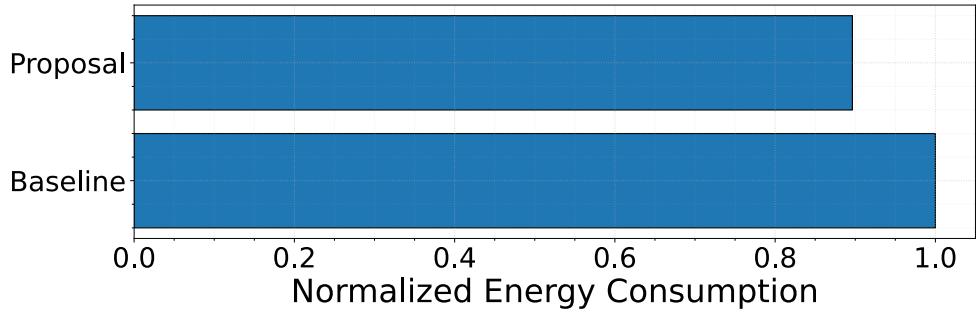


Figure 4-19: Comparison of the normalized PL energy consumed during workload execution between the workload optimization methodology and the baseline approach.

The results show that minimizing the interaction between tasks also has a significant impact on energy consumption, resulting in an energy reduction of 17.21% and 10.35% for the PS and PL, respectively.

Note that this energy reduction comes mainly from the makespan speedup and not from a reduction in terms of the power consumed by the test platform, which is the reason why this workload optimization methodology has been focused primarily on makespan optimization. However, power consumption insights, although less relevant than makespan insights, can be utilized within a higher-level scheduling

methodology. This approach leverages power consumption characterization to inform platform selection decisions at run time, given the significant variations in power consumption between platforms.[6]

Fairness Evaluation

The α configuration parameter enables steering the optimization goal between the two proposed targets, makespan and fairness. This section explores the impact of α on the optimization results by executing the synthetic workload multiple times on the hardware platform, keeping every scheduler parameter fixed but α, which is evaluated in the range: $[0.9, 0.5, 0.1]$. The baseline FCFS approach is included as a reference.

Figure 4-20 and Figure 4-21 show the evolution of the workload makespan and wait time, respectively, when varying the α parameter. The FCFS reference approach is represented as the red line.

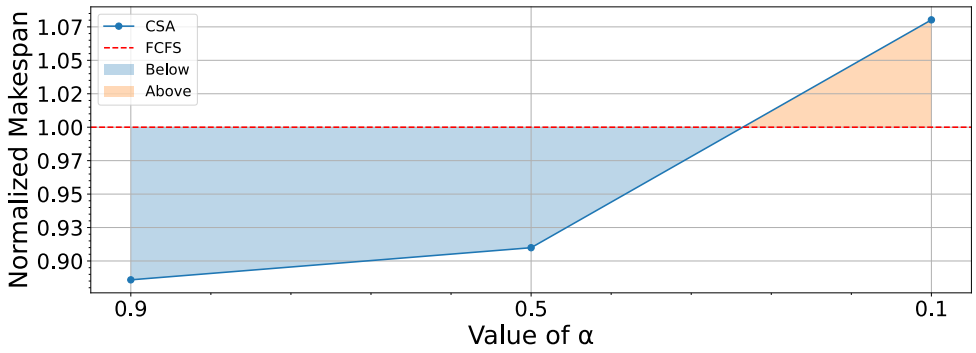


Figure 4-20: Evolution of the makespan based on the value of α (normalized to the reference).

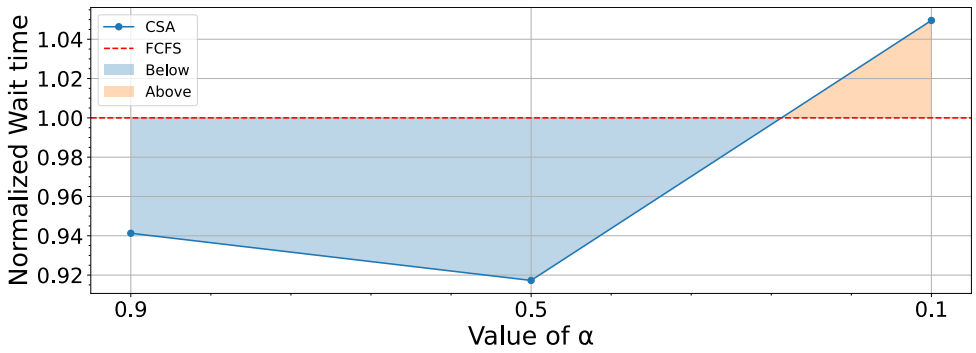


Figure 4-21: Evolution of the wait time based on the value of α (normalized to the reference).

It can be observed in Figure 4-20 that decreasing α results in a higher workload makespan and indirectly higher energy consumption. These findings are consistent with the analysis made in the sensitivity analysis, since a higher focus on fairness results in more tasks running in parallel, increasing their interaction, and

[6]The higher-level scheduler is planned as future work.

consequently, the overall workload makespan; even to the point where the $\alpha = 0.1$ case is worse than the FCFS approach.

Alternatively, Figure 4-21 shows that higher values of α (e.g., $\alpha = 0.9$) render wait times worse than the baseline since not promoting fairness at all could lead to certain tasks not being executed, which increases the overall wait time. Decreasing the α value (e.g., α=0.5) improves the wait time, as FPGA time is balanced between tasks. However, an excessive amount of fairness (e.g., $\alpha = 0.1$) can result in the opposite outcome. The reason is that, to excel in fairness, the FPGA needs to be shared between as many tasks as possible simultaneously, leading to a phenomenon of task interaction so significant that it is counterproductive.

To support these conclusions, in addition to the sensitivity analysis performed, the actual history of task scheduling decisions made is represented, per α value, in Figure 4-22, Figure 4-23, Figure 4-24, and Figure 4-25. These graphs show the distributions of the scheduled tasks per reconfigurable slot of the target FPGA over time, depending on the selected value of α. Since it is not possible to capture the very same moment in each run (due to run-time dynamicity and distinct scheduling strategy based on α), the plots, for the sake of analysis, are snapshots of an arbitrary time window representative of its corresponding α value. Therefore, the tasks scheduled might differ, and the time axis does not show an absolute value. However, the represented decisions are actual scheduling decisions that occur in hardware, and the time windows represent the same amount of time.

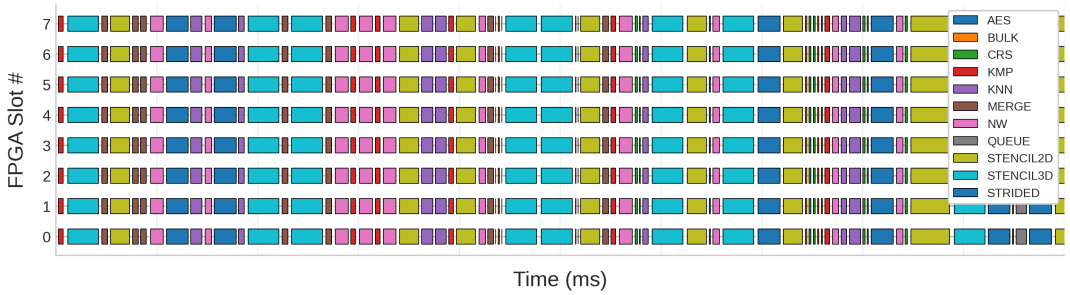


Figure 4-22: Representation of the scheduler decision strategy when $\alpha = 0.9$.

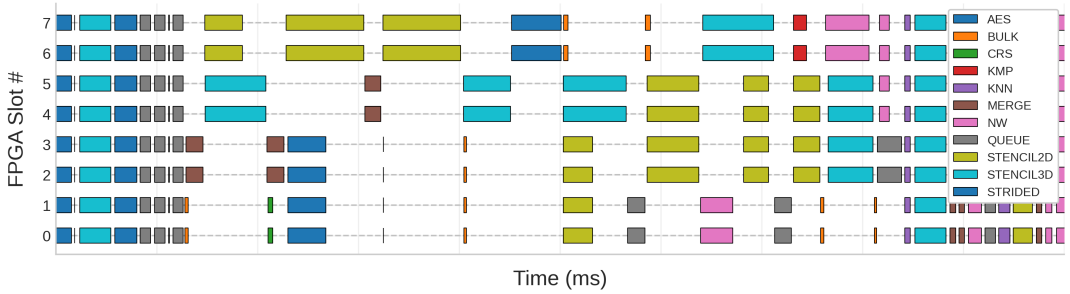


Figure 4-23: Representation of the scheduler decision strategy when $\alpha = 0.5$.

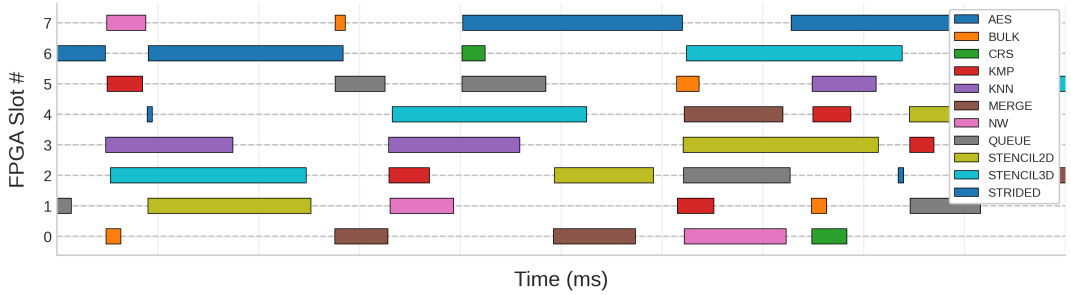


Figure 4-24: Representation of the scheduler decision strategy when $\alpha = 0.1$.

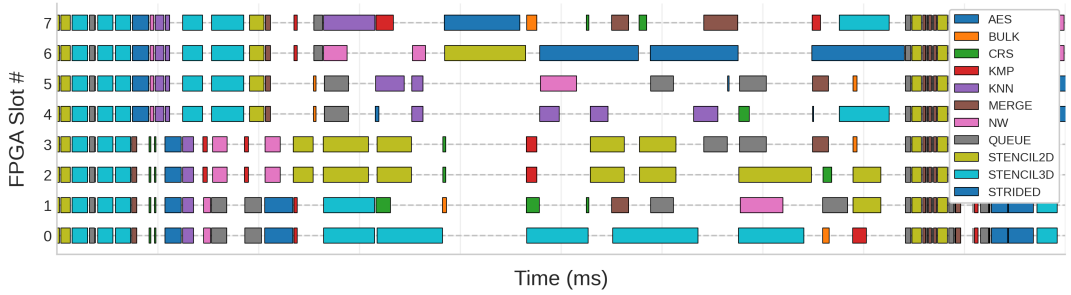


Figure 4-25: Representation of the scheduler decision strategy for the baseline FCFS.

As it can be observed, when $\alpha = 0.9$ (Figure 4-22), the scheduler avoids co-scheduling tasks, eliminating the task interaction. This behavior explains its lower makespan but high wait time. In turn, an intermediate $\alpha = 0.5$ (Figure 4-23) makes the scheduler place several tasks in parallel, which increases makespan (due to task interaction) but reduces wait time. Finally, $\alpha = 0.1$ (Figure 4-24) maximizes fairness by scheduling multiple tasks in parallel, which introduces a task interaction so high that it produces the worst makespan and wait time. The FCFS baseline (Figure 4-25) lies between $\alpha = 0.5$ (Figure 4-23) and $\alpha = 0.1$ (Figure 4-24). Its makespan and wait time are in the middle of them, since the number of tasks scheduled in parallel (i.e., task interaction) is intermediate.

4.4.6 Portability of the Workload Optimization Methodology

As it has been discussed throughout this document, portability and generalizability are fundamental design principles behind the entire work proposed in this Thesis. These principles ensure that solutions are not tightly coupled to a specific hardware setup, enabling deployment across a broad spectrum of reconfigurable platforms typically found in the cloud-edge continuum.

Following the same rationale, this section evaluates the proposed scheduling strategy when applied to new target platforms, the Pynq-Z1,[7] and Alveo U250.[8]

[7] XC7Z020-1CLG400C device.
[8] A-U250-P64G-PQ-G device.

These platforms differ significantly from the ZYNQ UltraScale+ ZCU102 in terms of computational capabilities and available logic resources. As it has already been discussed in previous chapters, the Pynq-Z1 board is a lower-end edge device with limited resources, while the Alveo U250 is a cloud-grade device that features a vast amount of resources.[9] The experiments replicate the same workload and scheduling procedures described earlier, now constrained to a maximum of 4 and 16 parallel accelerators for the Pynq-Z1 and Alveo U250 platforms, respectively.

The results obtained on each platform are presented in Table 4-6 and Table 4-7 for makespan and energy consumption, respectively. These tables show each platform's metrics side by side to ease comparison. In the case of makespan, the metrics presented are the breakdown of the workload makespan when using the proposed workload optimization methodology compared to the baseline FCFS approach, each value normalized to the overall workload makespan of the baseline approach. The same method is followed for the energy consumption, but the metrics are expressed per power rail available in the target platform.

Table 4-6: Comparison of the induced overhead and prediction error per scheduling approach on every platform (smaller is better).

Scheduling Approach	Overall Workload Makespan (%)			Workload Makespan Breakdown								
				Workload Execution (%)			Scheduling (%)			Modeling (%)		
	Z	P	A	Z	P	A	Z	P	A	Z	P	A
Baseline	100	100	100	99.98	99.99	99.99	0.02	0.01	0.01	–	–	–
Proposal	88.59	83.14	89.92	82.15	78.90	89.30	2.79	1.62	0.03	3.65	2.62	0.59

Z: Stands for the Zynq UltraScale+ ZCU102 board (i.e., main experiments)
P: Stands for the Pynq-Z1 board (i.e., portability experiments)
A: Stands for the Alveo U250 board (i.e., portability experiments)

Table 4-7: Comparison of the energy consumption per scheduling approach on every platform.

Scheduling Approach	Energy Consumption Breakdown			
	Z		P	A
	PS	PL	Board	Board[1]
Baseline	100%	100%	100%	100%
Proposal	82.78%	89.64%	83.32%	89.92%

[1] Just the FPGA board, the host CPU has not been measured
Z: Stands for the Zynq UltraScale+ ZCU102 board (i.e., main)
P: Stands for the Pynq-Z1 board (i.e., portability experiments)
A: Stands for the Alveo U250 board (i.e., portability experiments)

[9] For a deeper portability and generalizability analysis of the other components of this Thesis, please refer to Section 3.7.3.

It can be observed that, despite the limitations of the Pynq-Z1 platform, the scheduling strategy performs similarly to the ZYNQ UltraScale+ ZCU102, demonstrating robustness across platforms. In fact, the results show an even higher reduction in makespan in this case, reducing the overall workload to 16.86% compared to the baseline. The reason behind this improvement lies in the fact that running a workload in this more limited platform (fewer reconfigurable slots and CPU cores) extends the task execution time, reducing the relative impact of the scheduling and modeling processes, making workload optimization more significant. Similarly, improvement in energy consumption is also achieved in this lower-end platform.

The results are also similar for the Alveo U250 board, showing a reduction in workload execution time of 10.08% relative to the baseline. However, the observed optimization gain is slightly lower than in the other cases. These results suggest that the overall impact of kernel interaction in the Alveo scenario is lower than in the others; hence, minimizing it translates into less significant optimizations (still over 10%). The most likely explanation is that the powerful host CPUs in the deployment with the Alveo board makes CPU contention, which has a significant impact on the overall contention, less noticeable. It is also interesting that the high-performance CPUs used as a host on the Alveo U250 platform makes the scheduling procedure almost negligible in terms of overhead.

It must be noted that the same scheduler configuration has been reused without platform-specific tuning, supporting the generalizability of the approach. These results confirm that the proposed scheduler can operate effectively on lower- and higher-end devices, requiring minimal adaptation while maintaining its ability to exploit run-time characterization and conflict-aware scheduling decisions.

4.5 Conclusions

This chapter has presented a run-time workload optimization methodology that uses a metaheuristic-based scheduling algorithm, aided by data-driven workload characterization mechanisms, to offer multi-objective workload optimizations (i.e., makespan and fair use of resources) in the long run. In addition, a sensitivity analysis has been conducted to explore the impact of the scheduler parameters on the optimization results.

The validation campaigns presented in this chapter highlight two key contributions of the proposed workload optimization methodology and support its practical applicability in dynamic, heterogeneous environments like the cloud-edge continuum.

First, the proposed conflict-aware scheduling strategy, guided by run-time workload characterization and implemented using a lightweight metaheuristic (CSA), significantly improves workload execution on reconfigurable systems. Unlike static

approaches, this methodology adapts to dynamic conditions by capturing and leveraging real-time information about task interactions and system state. As a result, it achieves significant reductions in workload makespan even when accounting for the overhead of modeling and optimization. These improvements are particularly relevant in long-running workloads, where the benefits of adaptive scheduling are amplified. Moreover, the reduction in execution time directly translates into energy savings, a must in energy-constrained edge deployments. Alternatively, the user can steer the optimization goal towards fairness, enabling lower wait times if its needs require it.

Second, the experiments demonstrate that the proposed methodology is portable and generalizable. Without any platform-specific tuning, the same scheduling approach has been successfully ported from a mid-end FPGA device onto lower- and higher-end alternatives, maintaining its performance and energy benefits. This confirms the robustness of the solution and its suitability for a wide range of deployment scenarios, from powerful cloud nodes to resource-constrained edge devices. Such versatility is critical for supporting workload optimization across the whole platform spectrum of the cloud-edge continuum.

Together, these findings validate the effectiveness of combining incremental data-driven modeling with conflict-aware scheduling to manage dynamic workloads in reconfigurable multi-accelerator systems efficiently.

Chapter

5 COMPONENT INTEGRATION

This chapter presents the integration of the components developed in this Thesis: the workload management infrastructure, the run-time workload characterization, and the workload optimization methodology. To this end, a two-cluster cloud-edge continuum architecture is adopted, comprising multiple FPGA-based platforms that represent the different layers of the continuum (i.e., cloud, fog, and edge). The proposed computing architecture is validated through two complementary use-case scenarios. The first employs benchmark-based synthetic workloads to push the system to its limits, enabling the assessment of its portability, scalability, and optimization capabilities as well as the exploration of trade-offs. The second deploys a real-time Unmanned Aerial Vehicle (UAV) image-to-map matching application, leveraging per-layer kernel specialization and collaborative execution between nodes, enabling the exploration of performance and energy trade-offs in a real-world application scenario.

The chapter begins by introducing the objectives and scope, followed by a description of the requirements for integrating the components. It then presents a detailed description of the cloud-edge continuum architecture that is deployed and evaluates the two use-case scenarios, before providing an overall comparison of the results. The chapter concludes by summarizing the key findings and lessons learned.

5.1 Chapter Overview

Although the individual components developed in this Thesis have been thoroughly described and validated in their respective chapters, their full potential emerges when they are integrated into a cohesive and distributed computing environment. In this context, integration refers to the deployment and coordinated operation of all these components within a unified architecture, where they interoperate seamlessly to execute workloads, monitor and characterize them at run time, and optimize their execution dynamically across heterogeneous FPGA-based resources.

Achieving such integration in the cloud-edge continuum is not trivial. It requires not only ensuring that each component operates correctly in isolation, but also enabling their cooperation across layers and nodes with different capabilities, constraints, and execution contexts. This implies addressing multiple challenges

already discussed in this manuscript: managing a diverse set of FPGA-based platforms; enabling seamless application migration across geographically distributed nodes; and performing local optimization of workloads whose characteristics, arrival patterns, and kernel interactions vary unpredictably.

Without this cohesive integration of components, operating such an environment would require significant manual intervention, including handling platform-specific deployment of hardware configurations, manually transferring data between distributed systems, and making scheduling decisions individually for every device. Such a fragmented approach would be error-prone, inefficient, and not capable of reacting promptly to workload changes, ultimately leading to performance degradation and poor resource utilization.

To demonstrate the benefits of the proposed integration, a multi-cluster cloud-edge continuum architecture with two Liqo-connected Kubernetes clusters has been implemented. Each cluster contains FPGA-based platforms covering all continuum layers: a cloud-grade node with vast reconfigurable resources, a fog node with moderate resources, and four edge nodes optimized for low-power operation. Within this architecture, the Thesis components operate together to provide autonomous hardware accelerator deployment, continuous workload monitoring, local run-time characterization, and conflict-aware optimization across all nodes and clusters.

The effectiveness of this integration is validated through two complementary use-case scenarios. The first employs benchmark-based synthetic workloads to stress the system, evaluate its scalability, portability, and optimization capabilities, and explore the available trade-offs. The second involves a real-world UAV image-to-map application, demonstrating the applicability of this work in a practical scenario. Together, these use cases confirm both the technical robustness of the solution and its operational advantages in realistic distributed FPGA-based cloud-edge environments.

Figure 5-1 provides a visual synthesis of all the components developed in this Thesis, integrated into the multi-cluster cloud-edge continuum architecture. Unlike in previous chapters, where only the component under discussion was highlighted, here every element is emphasized: the workload management infrastructure (in blue), the run-time characterization (in green), and the workload optimization (in orange). These components are replicated and arranged to resemble the physical layout of the deployed architecture, with cloud-, fog-, and edge-grade FPGA nodes connected within and across clusters. This integral view illustrates the interoperation of the individual components and their joint role within the complete system.

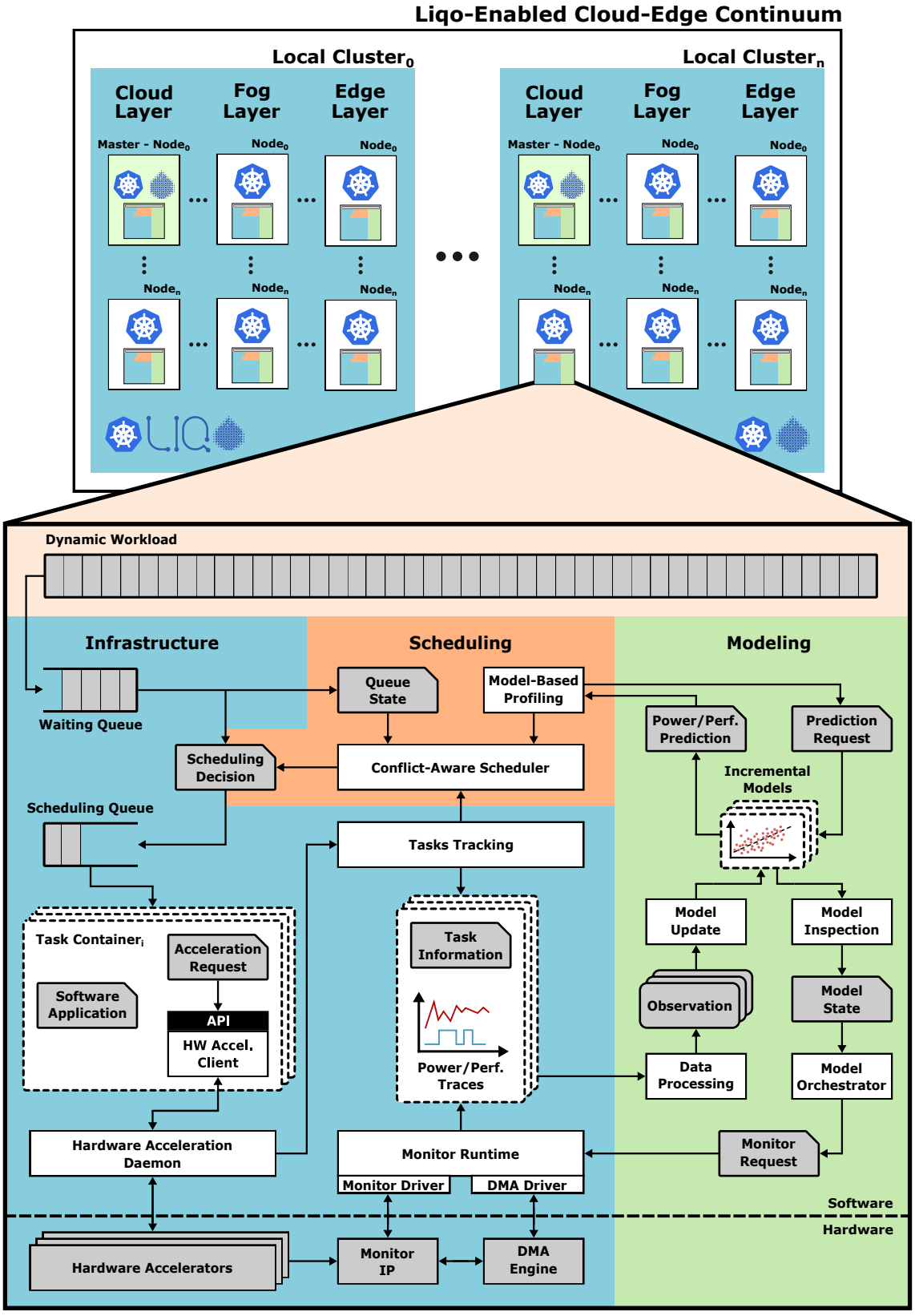


Figure 5-1: Details of the proposed multi-cluster cloud-edge continuum.

5.2 Integration Requirements

As it has been emphasized throughout this Thesis, using FPGAs in the cloud-edge continuum is not straightforward. It requires meeting a wide range of technical and operational requirements to make the approach both viable and efficient in practice. These requirements arise from the combination of heterogeneous architectures, geographically distributed deployments, diverse application demands, and the need for prompt adaptability to dynamic workloads. While these requirements (described in previous chapters) are individually addressed by the components developed in this Thesis, fully meeting them demands all components to be integrated into a single cloud-edge continuum architecture.

As a foundation of this Thesis, portability and generalizability have been prioritized to ensure that the proposed methodologies can be applied across a wide range of platforms, environments, and domains beyond the specific experimental setups used for validation. This design philosophy, in which the components have been developed and validated incrementally (as shown in each chapter) with interoperability as a guiding principle, makes the final integration straightforward and avoids the need for major re-engineering.

The remainder of this section is organized as follows. First, the node-level integration, which addresses the requirements for deploying and optimizing workloads within a single FPGA-based system, is presented. Second, the continuum-level integration is described, covering the management of multiple nodes across the cloud-edge continuum, including multi-cluster federation. Each part includes a detailed mapping of requirements to the specific components that address them.

Node-Level Requirements

At the node level, the goal is to ensure that each FPGA-based system can autonomously receive workloads, configure its hardware accordingly, monitor its performance and power consumption, and continuously optimize its execution based on the current conditions. Table 5-1 summarizes the main requirements at this level, along with the components responsible for addressing them.

Table 5-1: Node-level requirements and components addressing them.

Requirement	Responsible Component(s)	Chapter
Autonomous workload reception and deployment	Workload Management Infrastructure	Chapter 2
Platform-independent hardware acceleration	Workload Management Infrastructure	Chapter 2
Synchronized monitoring of power and performance	Workload Management Infrastructure	Chapter 2
Dynamic characterization of workloads	Run-time Workload Characterization	Chapter 3
Conflict-aware workload optimization	Workload Optimization Methodology	Chapter 4

The workload management infrastructure (Chapter 2) abstracts platform-specific hardware acceleration details, ensuring that the correct configuration is deployed regardless of the underlying FPGA architecture. This is supported by the execution model and virtualization mechanisms described in Section 2.3.2, which enable portability of acceleration tasks across heterogeneous nodes. The monitoring framework (Section 2.3.3) acquires synchronized power and performance traces during execution, which are then processed by the run-time workload characterization (Chapter 3) to dynamically update predictions. These predictions feed the workload optimization methodology (Chapter 4), which selects the optimal set of tasks and replicas to execute, while mitigating conflicts between accelerators running in parallel.

Since these components were developed incrementally, with the optimization methodology building on the characterization models and these in turn relying on the workload management infrastructure, they fit together seamlessly, without additional engineering effort. As a result, each FPGA node operates as a self-adaptive computing element within the continuum, capable of responding locally to changing workload and environmental conditions. This autonomy avoids the need for centralized coordination or manual intervention, making scalable deployment in the cloud-edge continuum feasible.

Cloud-Edge Continuum Requirements

At the continuum level, the challenge is to coordinate multiple FPGA-based nodes distributed across the different layers of the cloud-edge continuum (and potentially across different administration entities) while preserving autonomy, scalability, and performance isolation. Table 5-2 lists the main requirements at this level and the components that address them.

Table 5-2: Continuum-level requirements and components addressing them.

Requirement	Responsible Component(s)	Chapter
Portable deployment across heterogeneous nodes	Containerization + Kubernetes	Chapter 2
Coordination of geographically distributed clusters	Liqo-based Federation	Chapter 2
Preservation of autonomy, ownership, and security	Liqo-based Federation	Chapter 2

In this context, containerized acceleration tasks integrate seamlessly with Kubernetes, enabling portable deployment and orchestration across heterogeneous nodes without requiring modifications to the applications. This portability extends transparently to multi-cluster scenarios through Liqo-based federation, which allows geographically distributed clusters to share resources while maintaining autonomy, ownership, and security. These capabilities ensure that workloads can be scheduled across the entire cloud-edge continuum with minimal overhead. Meanwhile, local scheduling policies at each node maintain performance isolation and prevent centralized bottlenecks, thereby supporting a scalable and resilient infrastructure.

5.3 Integration Setup

This section describes the integration details of the two-cluster architecture proposed as the experimental setup for the posterior validation campaigns.

The experimental setup consists of two Kubernetes clusters connected via Liqo. Each cluster contains nodes corresponding to the three layers of the continuum: one cloud-layer node featuring a vast resource FPGA, one fog-layer node with moderate resources, and four edge-level nodes optimized for low-power operation. Table 5-3 details the selected platforms for each of the two equivalent clusters.

Table 5-3: Cloud-edge continuum experimental setup (per cluster).

Layer	Board	CPU Freq.	FPGA Freq.	# Reconf. Slots (LUTs/Slot)	# Instances
Cloud	Alveo U250[1]	2 GHz	100 MHz	16 (44.1K)	1
Fog	Zynq US+ ZCU102[2]	1.2 GHz	100 MHz	8 (21.6K)	1
Edge	Pynq-Z1[3]	650 MHz	100 MHz	4 (6.4K)	4

[1] A-U250-P64G-PQ-G device. [2] XCZU9EG-2FFVB1156 device. [3] XC7Z020-1CLG400C device.

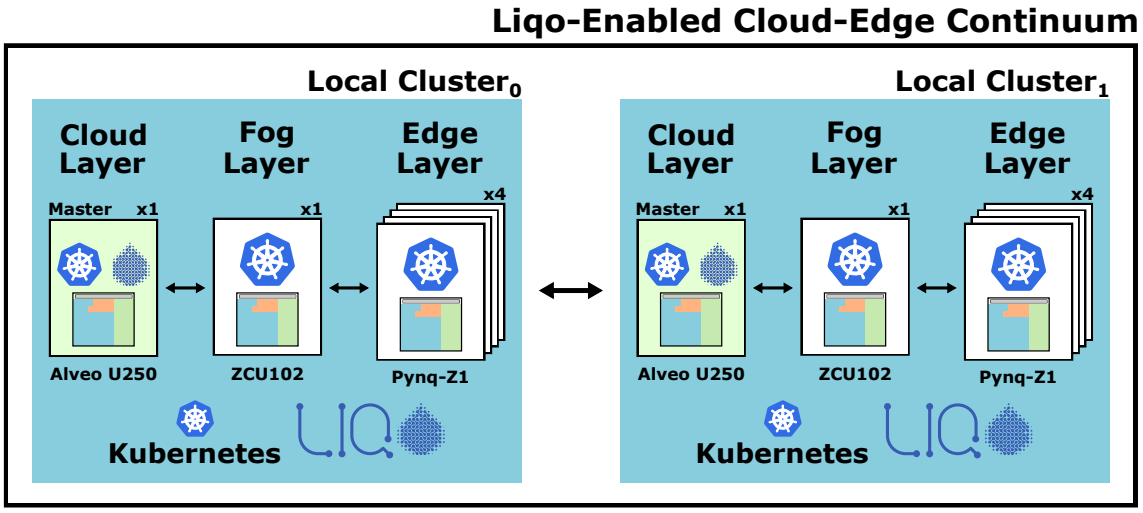


Figure 5-2: Representation of the two-cluster architecture used for validation. The master nodes that run the Liqo processes are highlighted in green.

A schematic representation of this two-cluster architecture is shown in Figure 5-2. It depicts the composition of the two mentioned clusters, each containing the platforms described in Table 5-3. Please note that both clusters are connected by relying on Liqo and Kubernetes, as was described in Section 2.4.2.

Table 5-4 enumerates a diverse set of clusters' board arrangements, which are the deployment scenarios used to evaluate the proposed cloud-edge continuum architecture in the two use cases. Please note that these deployment scenarios

include single-board, single-cluster single-layer, multi-cluster single-layer, single-cluster multi-layer, and multi-cluster multi-layer configurations, enabling a rich exploration of the available trade-offs. For the sake of clarity, any other inter-layer arrangements, while supported, have not been studied.

Table 5-4: Deployment scenarios of the experimental validation.

Scenario	# Boards	# Total Boards
Single Edge Node	1 Edge	1
One-Cluster Edge Layer	4 Edge	4
Two-Cluster Edge Layer	8 Edge	8
One-Cluster Fog Layer	1 Fog	1
Two-Cluster Fog Layer	2 Fog	2
One-Cluster Cloud Layer	1 Cloud	1
Two-Cluster Cloud Layer	2 Cloud	2
One Cluster	1 Cloud + 1 Fog + 4 Edge	6
Two Clusters	2 Cloud + 2 Fog + 8 Edge	12

5.4 Experimental Results

This section evaluates the integrated solution through two complementary use cases deployed onto the described setup. Each use case aims at a specific evaluation goal.

5.4.1 Use Case 1: Benchmark-Based Synthetic Workload

The purpose of this first use case is to stress the system to its limits and evaluate the possibilities that a multi-cluster architecture, managed and optimized using the methodologies proposed in this Thesis, can offer in terms of performance, energy efficiency, and scalability.

This validation campaign is conducted similarly to the ones presented in previous chapters. A dynamic workload composed of the MachSuite benchmarks described in Section 2.5.1 is executed on each of the deployment scenarios listed in Table 5-4. In this case, since several of the scenarios feature multiple computing nodes, the workload is made up of 4× more kernels than in previous validation campaigns. Therefore, 240,000

hardware acceleration requests are included in the workload, gradually incorporating additional kernel functions over time: the first 80,000 hardware acceleration requests feature only 4 kernels, the following 80,000 include another 4 additional kernels for a total of 8, and the last 80,000 include the 11 available kernels.

Since the workload optimization process in this Thesis operates at the node level, the continuum-level workload used for evaluation is manually partitioned into subsets, with each subgroup assigned to one of the nodes of the continuum. This approach requires careful partitioning so that every node in the scenario receives an amount of kernels that results in similar execution times. However, this manual process is tedious and error-prone, and in some deployment scenarios, especially in mixed-layer arrangements, some nodes still finish slightly faster than others. As future work, this process could be improved with an intelligent workload distribution mechanism operating at run-time at the continuum level to keep each node properly supplied with tasks.

The results of this use case are evaluated first as a comparison between scenarios featuring single boards. Afterwards, single-layer scenarios are compared, scaling the number of boards, followed by the comparison of multi-layer scenarios.

Single Board Results and Discussion

Figure 5-3 presents the time it takes to execute the workload on a single board of each layer. The results show both the execution of the baseline, non-optimized workload where tasks are executed in an FCFS fashion, and the execution time when leveraging the proposed workload optimization mechanism from Chapter 4 (i.e., CSA-based workload optimization). Please note that all the results are normalized to the fastest execution (i.e., the optimized cloud scenario).

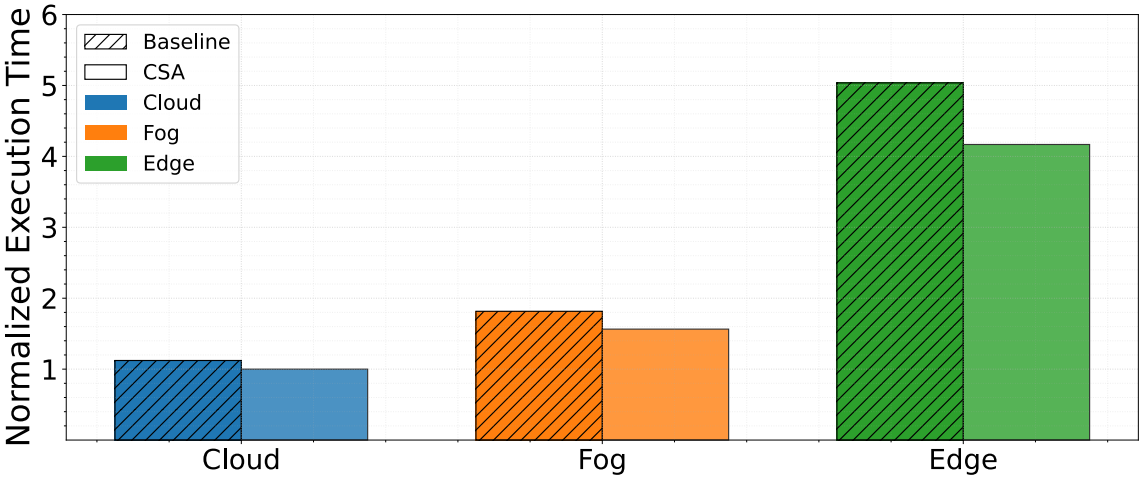


Figure 5-3: Workload execution time per layer, normalized to the fastest scenario.

As one would expect, the results show execution time increases as the computation is moved closer to the edge, since edge nodes have more limited computing resources.

It can be observed that the progression matches closely the results shown in the cloud-edge continuum validation of Chapter 2. The results, in terms of workload optimization's impact on execution time, are also similar to those presented in Chapter 4, with slightly higher improvements, as longer-lasting workloads dilute the impact of the workload characterization process.

In terms of power consumption, Figure 5-4 shows the power consumption per layer of a single board, normalized to the least power-demanding scenario (i.e., the edge scenario). It represents the consumption of the entire system (i.e., FPGA + CPUs).

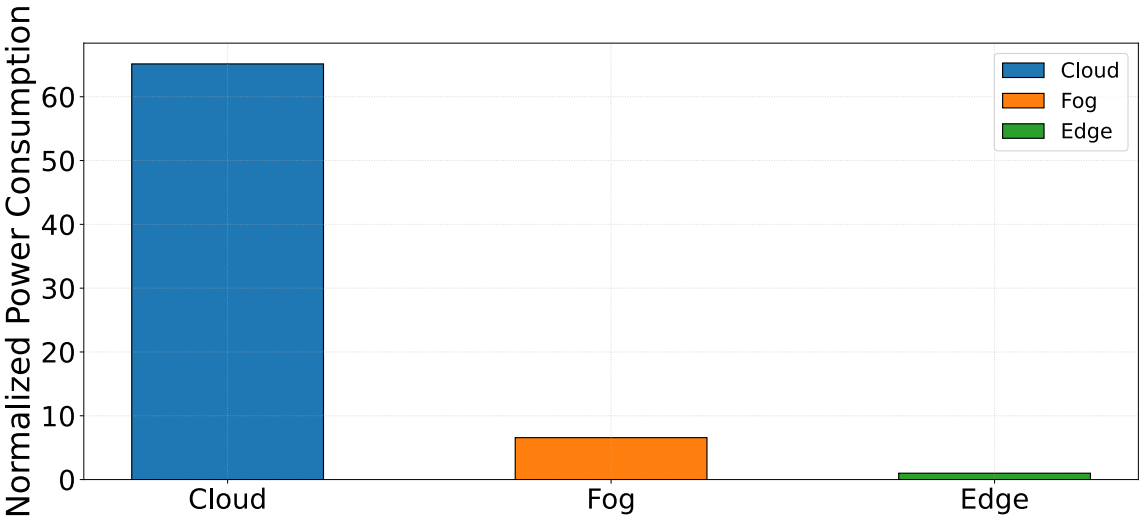


Figure 5-4: Power consumption per layer, normalized to the least-demanding scenario.

In this case, the power consumption increases as computation is moved towards the cloud, correlated with the amount of available resources. It is noteworthy that there is a 60× increase in power consumption when moving from the edge to the cloud.[1]

There is a clear trade-off between power consumption and performance when moving across the cloud-edge continuum, and even more can be explored when scaling the number of nodes, as the following sections analyze.

Layer-Level Results and Discussion

Scaling up the number of nodes present in a continuum layer to enable collaborative execution can have a significant impact on performance and power consumption.

Regarding performance, Figure 5-5 shows how the workload execution time varies as more nodes are included per layer, normalized to the scenario with the highest performance (i.e., the two-cluster cloud layer). Note that in the cloud and fog layers, one cluster features just one board, while at the edge, a cluster contains four nodes.

[1]Please note the CPU used for the cloud platform is an Intel Xeon Gold 6338, with a nominal power consumption of 205 W.

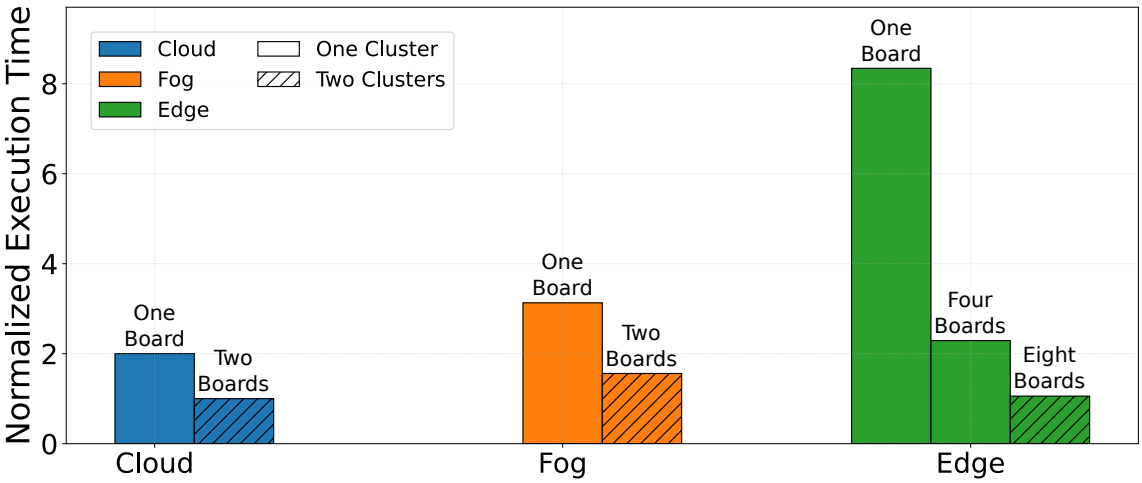


Figure 5-5: Workload execution time per layer and nodes, normalized to the fastest scenario.

Performance increases as more nodes of the same layer are introduced in the deployment, as expected. What is more interesting is that the performance of the two-cluster edge-layer scenario, which features 8 edge-grade boards, is quite similar to that of the two-cluster cloud-layer scenario, which features two cloud-grade platforms.

Figure 5-6, in turn, depicts the power variation as nodes scale up per layer. Again, it is normalized to the least power-hungry scenario (i.e., the single board at the edge).

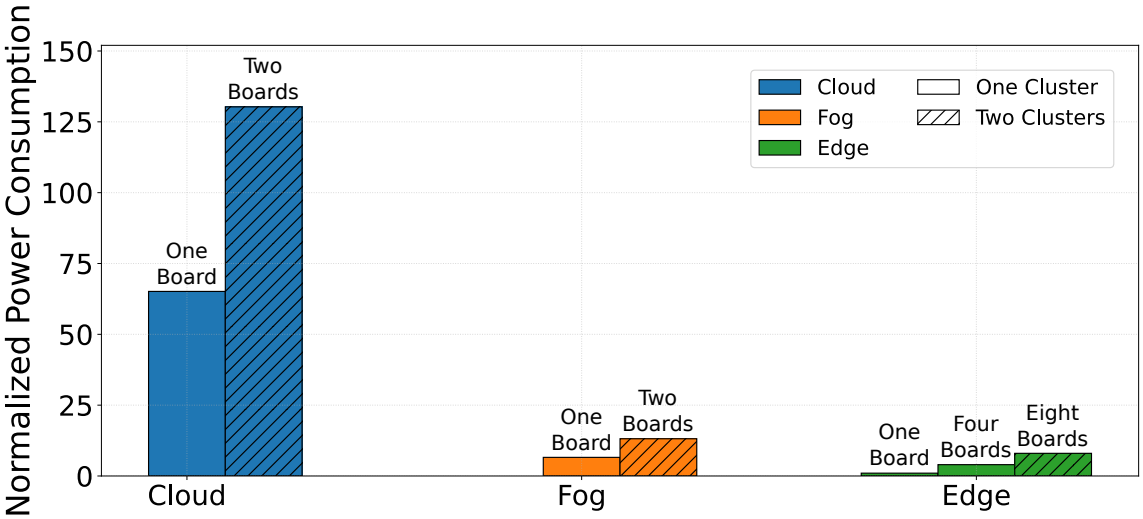


Figure 5-6: Power consumed per layer and nodes, normalized to the least demanding scenario.

Logically, power consumption increases with the number of boards, which skyrockets the power consumption of the two-cluster cloud layer scenario to more than 125× that of a single edge board.

As observed, the power/performance trade-offs can be exploited by both moving across the continuum and scaling up the number of nodes.

Continuum-Level Results and Discussion

To better analyze the trade-offs available in a cloud-edge continuum scenario, Figure 5-6 shows a scatter plot that represents the workload execution time and energy consumption of every deployment scenario listed in Table 5-4, including the single-cluster and two-cluster multi-layer scenarios composed of a mixture of board types. The graph shows the execution time on the horizontal axis, normalized to the highest-performing scenario (i.e., two clusters, multiple layers); meanwhile, the vertical axis presents the energy consumption per scenario, normalized to the most energy efficient (i.e., two clusters, edge layer). The size of the bubble, as well as the number in the center, indicates the total number of nodes in that scenario. In turn, the color highlights the type of boards that constitute the scenario. Finally, a thicker bubble edge represents a scenario that utilizes boards from both clusters, while the thinner edge represents single-cluster scenarios.

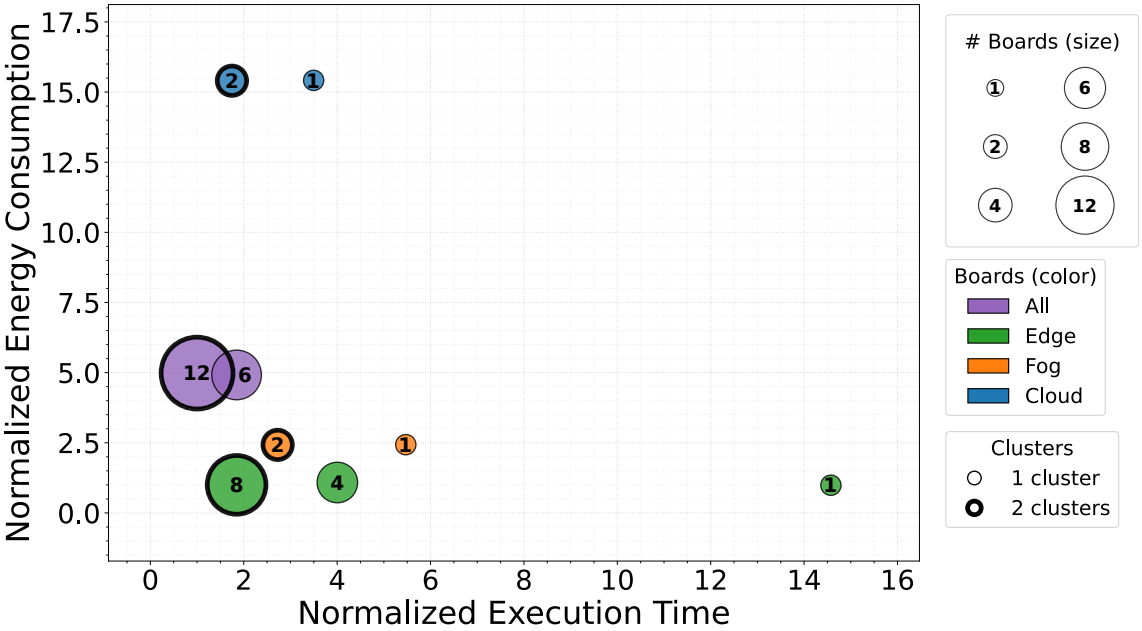


Figure 5-7: Workload execution time against energy consumption per layer and nodes, normalized to the best scenario in each axis. Bubble size indicates the number of boards. Color indicates board type. Thick edge means two-cluster scenario; thin edge, one-cluster scenario.

In terms of energy consumption, there is a clear conclusion to draw. In this particular scenario, processing at the edge is significantly less power-hungry than any other scenario. Then, as computation moves towards the cloud, energy consumption rises considerably. Somewhere in the middle, but close to the edge, lie the entire-cluster scenarios that use multiple boards of different types, combining their computing power but maintaining a reduced power consumption (compared to the cloud). Meanwhile, scaling up the number of nodes within a given layer does not impact energy, since the speedup compensates for the increase in power. Naturally, there may be occasions where the power budget does not allow for more parallel

solutions that, while reducing overall energy consumption, increase power demands.

Regarding execution time, it is observable that, again, moving from edge to cloud increases the performance of the scenario, at least in the single-board cases. Moreover, scaling up the number of nodes also results in a proportional reduction in execution time. Both full-cluster scenarios are the best performing, as they have, in essence, the highest amount of computing resources and hence can better exploit the parallelism derived from the collaborative workload execution.

It is noteworthy that the two-cluster edge layer scenario renders a performance comparable to the two-cluster cloud layer scenario and ahead of the fog layer. This is due to the fact that the two-cluster edge layer scenario features 8 edge nodes, each with four reconfigurable slots for a total of 32, the same as the ones in the two-cluster cloud layer scenario. Still, under an equal number of reconfigurable slots, having fewer nodes is beneficial in terms of performance, as each single board executes a larger workload, which reduces static overheads such as the one associated with the workload characterization process. Furthermore, as highlighted in Table 5-3, the amount of logic resources in the reconfigurable slots varies per layer. This benefit is not exploited by the MachSuite benchmarks, as they have been implemented equally across each platform; however, taking advantage of this, as will be shown in the subsequent use case, has a significant impact that could clearly favor the cloud in terms of performance.

Other trade-offs arise that can be exploited by users, such as the degree of privacy that increases as one approaches the edge, or latency, which also improves as one approaches the edge. In contrast, other criteria favor the cloud, such as scalability and redundancy, as resources are virtually unlimited in the cloud but not at the edge.

Keep in mind that many more node arrangements can be implemented by adjusting the number of boards per layer and the number of clusters. In fact, any position on the graph of Figure 5-7 can be potentially covered by a specific combination of diverse boards and clusters, allowing users to explore and exploit a vast landscape of trade-offs and possibilities. This exploration and exploitation is enabled by the different components, technologies and methodologies proposed in this Thesis.

5.4.2 Use Case 2: Feature Extraction and Matching for UAVs

The purpose of this second use case is to demonstrate how more elaborate user applications can exploit the possibilities offered by the proposed infrastructure in the cloud-edge continuum. To do so, a feature extraction and matching pipeline has been developed and evaluated with the proposed two-cluster architecture. The remainder of this section first introduces the concepts of feature extraction and matching, followed by a brief description of state-of-the-art approaches. Then, the implementation details of the selected algorithms are presented, and finally, a validation campaign is conducted to assess the possibilities of the proposed cloud-edge continuum architecture for such applications.

Motivation

Small UAVs often need precise position updates in places with a lack of satellite signals (i.e., along narrow streets with tall buildings, across valleys, or in areas with deliberate interference). In these scenarios, the ground becomes the only reliable reference: roads, roofs, and field boundaries form a stable visual aid that does not depend on external infrastructure. The challenge is to read that information quickly and reliably from a moving device with limited energy budgets and strict latency requirements.

A common approach to solving this problem relies on a downward-facing camera to provide a continuous stream of visual information, and feature extraction and matching algorithms to process that stream, identifying specific structures in the image that can be used to infer the UAV's position. The method is deliberately simple: convert each frame into a set of distinctive indicators (i.e., features), search for those indicators on a prepared map, and deduce the position when a consistent set of correlations is detected. This approach is both low-cost and scalable. It is also well-suited for the cloud-edge continuum, where multiple compute nodes can be leveraged when needed (even collaboratively), and for FPGA implementations, specifically at the edge, where low-power processing is particularly effective.

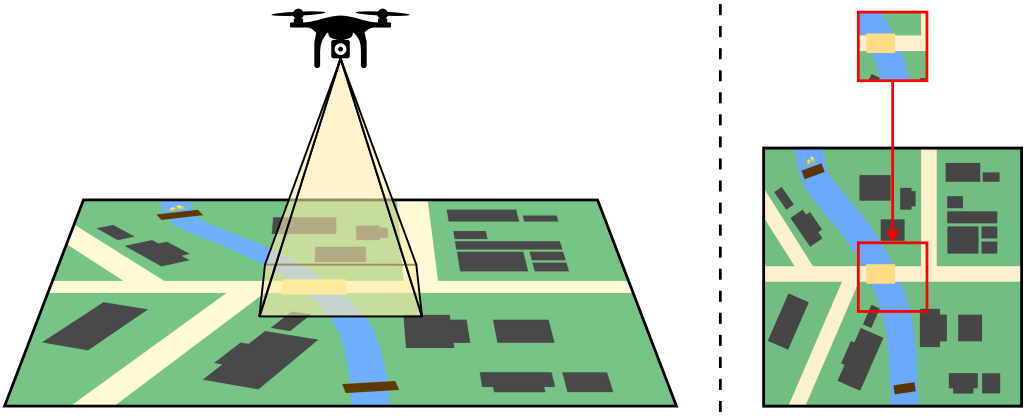


Figure 5-8: Representation of the UAV image-to-map application.

Background

A feature or keypoint is a small and distinctive image pattern, often a corner or irregular point, that remains present in different views of the same place. The primary idea of feature extraction and matching is to discover the same physical points across two images and use those connections to infer how the images relate to each other.

Figure 5-9 depicts a standard feature extraction and matching pipeline, which contains the next three stages:

- **Extraction:** keypoints are marked at locations whose appearance is distinctive and stable under moderate changes in perspective and illumination.

- **Description:** around each keypoint, a vector summarizes the local structure in a way that images from close locations lie close to each other in descriptor space while images from different locations remain far apart. Descriptors exist in float and binary representations, with binary codes being faster to compare and more efficient to transmit.
- **Matching:** descriptors from one image are paired to descriptors from another image by their distance in the descriptor space (e.g., Hamming for binary, Euclidean for float). Some mechanisms, such as bidirectional matching, are employed to prevent ambiguity.

Once a sufficient set of pairs is found, a planar mapping (i.e., homography) is performed to map one image to the other, as illustrated at the bottom of Figure 5-9.

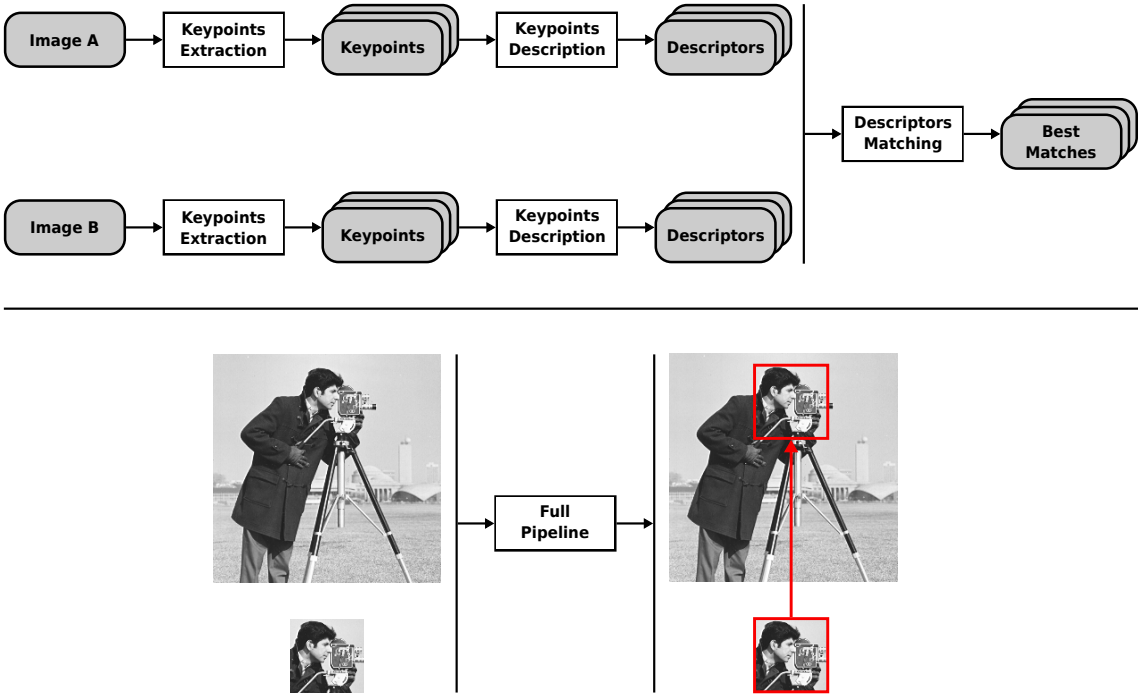


Figure 5-9: Feature extraction and matching pipeline.

State of the Art

This section briefly describes the main trends followed in the literature for each feature extraction and matching pipeline. Then, for each pipeline stage, the candidate most suitable for the target application scenario is selected.

Regarding keypoint extraction, corner-based detectors such as Harris [Harris'88] and Shi–Tomasi [Shi'94] identify keypoints using image gradients. They localize precisely and demonstrate good repeatability, but usually rely on floating-point arithmetic and substantial memory bandwidth. Scale-space detectors such as

SIFT [Lowe'99] and SURF [Bay'06] increase robustness by searching for distinctive structures, introducing scale and rotation invariance. This comes with broader kernels and heavier computing requirements. In contrast, FAST [Trajković'98] and its variants test a small ring of pixels around each candidate using ordered intensity comparisons. Most non-corners are rejected after a few tests, reporting exceptional speed and making these detectors well-suited to performance-limited implementations.

In terms of keypoint description, float-based descriptors like SIFT [Lowe'99] and SURF [Bay'06] are strong alternatives thanks to their high discriminative results, though they are heavier to compute, store, and compare. Binary descriptors such as BRIEF [Calonder'10], ORB [Rublee'11], BRISK [Leutenegger'11] or FREAK [Alahi'12] encode each keypoint description as a short bitstring formed from simple intensity comparisons at pre-selected point pairs. They are compact, efficient to transmit, and fast to compare with Hamming distance.[2] The trade-off is slightly reduced robustness under extreme viewpoints or illumination changes.

For keypoint descriptor matching, approximate nearest-neighbor approaches (e.g., FLANN [Muja'09], LSH [Gionis'99]) can speed up searching in extensive databases but add indexing overhead and some non-determinism. In turn, exact brute-force matching is deterministic and straightforward to parallelize: each descriptor is compared against candidates, resulting in a distance between them (e.g., Hamming distance). Cross-check further reduces outliers at modest cost.

Moreover, there are recent methods such as SuperPoint [DeTone'18] or SuperGlue [Sarlin'20] that significantly improve the matching under complex viewpoint and illumination changes by learning both description and matching from a previous dataset. They stand out when conditions are difficult (e.g., variations in illumination, rotation, scale), at the expense of heavier compute requirements and memory demands. These properties make them attractive in server-class environments.

In the proposed application scenario, where the targets are top-view aerial images with limited scale and rotation changes, the scene varies slowly and remains close to planar. That favors detectors and descriptors that are fast and deterministic rather than overengineered for extreme viewpoint changes. FAST meets this need: it finds repeatable keypoints using integer tests on a small neighborhood, ensuring high keypoint density without heavy processing. Combining FAST with a BRIEF-256 descriptor keeps the representation compact (256 bits per keypoint), minimizing memory and bandwidth, essential for on-board storage and edge-to-cloud transfers. Just as important, this choice fits clearly in FPGA fabrics. FAST local, ordered comparisons and BRIEF binary tests can be implemented as deeply pipelined, line-buffered kernels with simple control and high parallelism possibilities. Finally, Hamming comparisons map to XOR operations plus counting, enabling scalable throughput. Therefore, a brute-force matching approach based on the Hamming distance between descriptors is simple, deterministic, and easily parallelizable, enabling scaling across cloud-edge continuum nodes.

[2]The Hamming distance is the number of positions at which two equal-length vectors differ.

In summary, the proposed implementation involves three key components: FAST extraction for identifying keypoints from images, BRIEF-256 descriptors for computing an identification for each keypoint, and brute-force Hamming matching to find related keypoints. Such an approach will enable two structures to be matched in two different images, enabling the expected self-localization capabilities for the target application.

Implementation Details

The implementation of each stage of the feature extraction and matching pipeline has been carried out using HLS. This section provides a concise description of each algorithm implementation.

- **FAST extraction:** the FAST algorithm processes each frame pixel by pixel and declares a keypoint at pixel p when at least N contiguous samples (nine in this implementation) on a 16-pixel circle are either all brighter than I_p+t or all darker than I_p-t, with I_p being the intensity at pixel p and t a user-defined threshold (see Figure 5-10). This algorithm can lead to the extraction of multiple similar keypoints, which do not add more information, but require computing power; hence, a non-maximal suppression technique is implemented to keep only the most distinctive point in a local region. After the FAST algorithm, each keypoint has a strength computed by aggregating the intensity difference between p and the 16 surrounding pixels. The implemented non-max suppression analyzes 3×3 pixel regions; only the pixel with the highest difference value is kept in each local neighborhood. Moreover, to enforce homogeneous density, detections are capped to a maximum number per local region, preventing clusters of nearly identical points from dominating later stages.

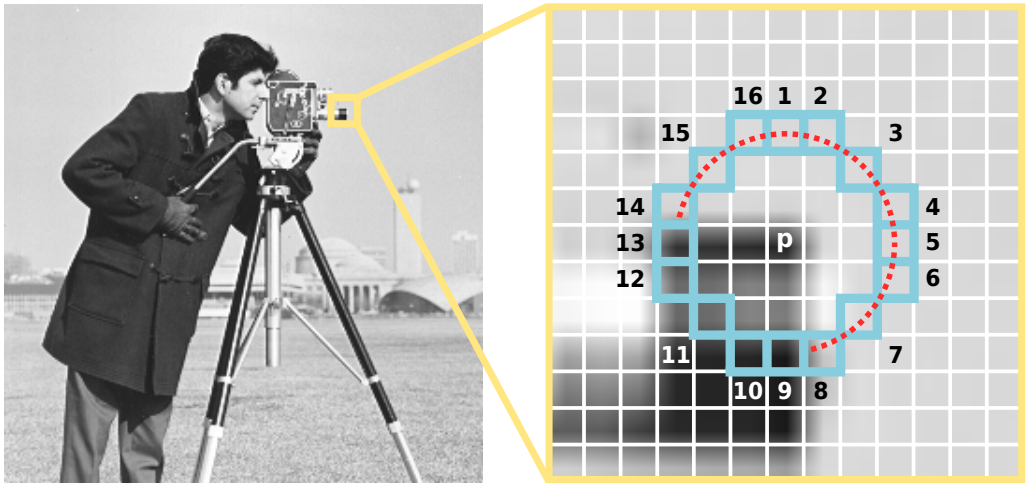


Figure 5-10: Representation of the FAST extraction algorithm.

- **BRIEF description:** each extracted keypoint is described using the BRIEF algorithm, represented in Figure 5-11, which encodes information from the surroundings of the keypoint to obtain a numerical fingerprint. For each detected keypoint, a 31×31 patch is extracted at the same image location where it was found, as shown in the figure (the keypoint is highlighted in red in the middle of the patch). A pattern, fixed at design time, of 256 point pairs is applied to the patch (blue, green and orange lines represent three of those 256 pairs). For each pair, a bit is extracted representing which of the two points of the pair is brighter. The 256 pairs generate 256 bits that are concatenated into a 256-bit descriptor that represents the keypoint (i.e., a numerical fingerprint). With this approach, the descriptor expresses the relation between the points surrounding a keypoint, and, since it is generated with a fixed pattern, similar keypoints result in similar descriptors in a reproducible manner.[3]

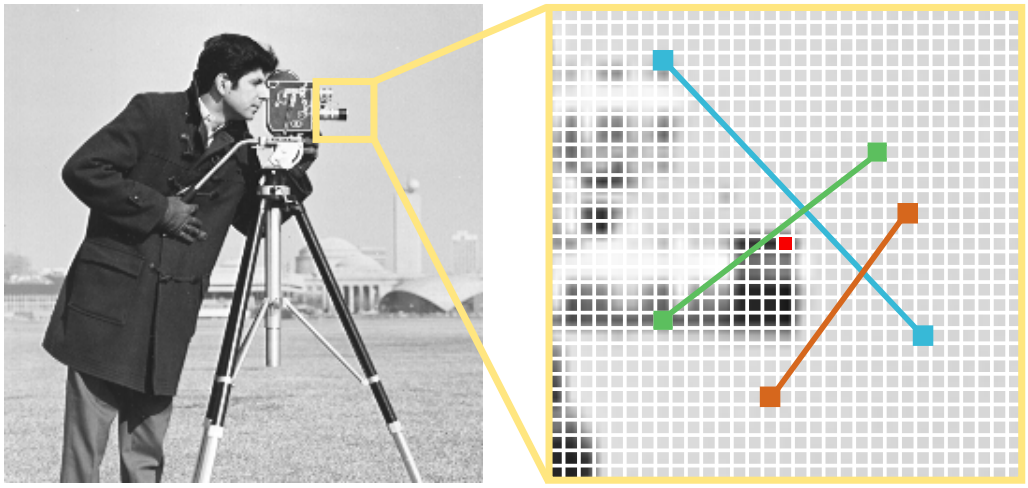


Figure 5-11: Representation of the BRIEF description algorithm.

- **Brute-Force matching:** descriptors from the one image are compared against descriptors from the other using their Hamming distance, following a brute-force approach. For each keypoint, the two nearest candidates are obtained. The best candidate is accepted only if its Hamming distance is sufficiently smaller than that of the second-best, avoiding ambiguous keypoints. As a result, the descriptors of one image are assigned to the descriptors of the other, as shown in Figure 5-12. Then, that mapping can be translated back into the keypoints represented by those descriptors.

[3] In this Thesis, the FAST and BRIEF algorithms have been designed to receive 96×96 tiles. Therefore, larger images are divided into 96×96-size tiles before processing starts.

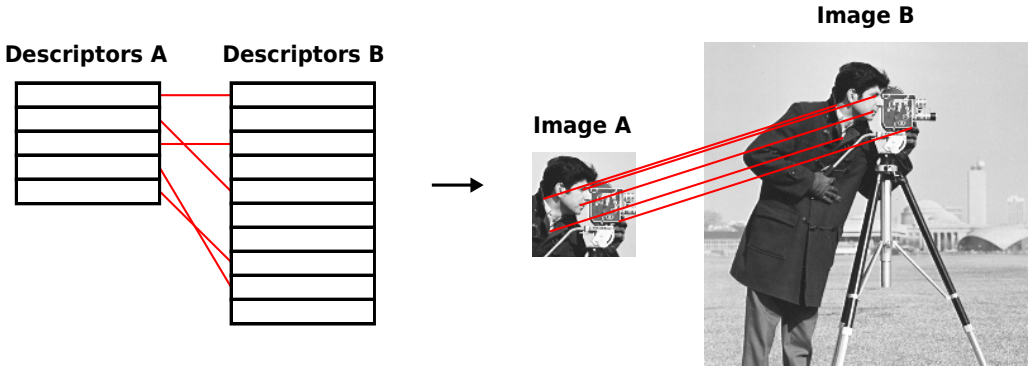


Figure 5-12: Representation of the Brute-Force matching algorithm.

The matches obtained indicate correspondences between the images' keypoints. However, the actual image-to-image map is obtained by estimating a homography (i.e., planar mapping) with the RANSAC algorithm [Fischler'81]. This homography estimation approach is fundamentally sequential and, as observed in Table 5-5, presents a negligible latency compared to the rest of the pipeline; therefore, it has been implemented in software.

Table 5-5: Software profiling of each pipeline stage (matching two 96×96 tiles).

Stage	FAST	BRIEF	BF Matcher	RANSAC
Latency (ms)[1]	3.921	7.251	25.197	0.312

[1] Measured in the 650 MHz dual-core ARM Cortex-A9 of the edge board

Experimental Setup

The validation campaign of this use case differs slightly from the other evaluations conducted in this Thesis. Instead of executing a dynamic workload with a heterogeneous set of single-kernel applications (as it has been evaluated in the previous use case), this time the workload is a single application with 3 different kernels, the feature extraction and matching pipeline depicted in Figure 5-9. The experiments consist of this application running continuously while fed with a video extracted from the downward-facing camera of a UAV. Given the simpler complexity of workload deployment (in terms of scheduling), the workload characterization and optimization components have been left out for this use case. In turn, the focus is on the possibilities offered by the proposed cloud-edge continuum architecture, due to the flexibility and scalability provided by the workload management infrastructure.

The experiments will consist of a top-view video of the ground seen from the camera, which will be processed (frame by frame) by the first two kernels of the pipeline, extracting representative keypoints and describing them. These descriptors are then evaluated against the pregenerated descriptors of a map representing the

terrain traversed by the UAV, employing the last kernel of the pipeline. A representation of this process is shown in Figure 5-13. Please note that the map shown in this figure has been cropped for the convenience of representation.

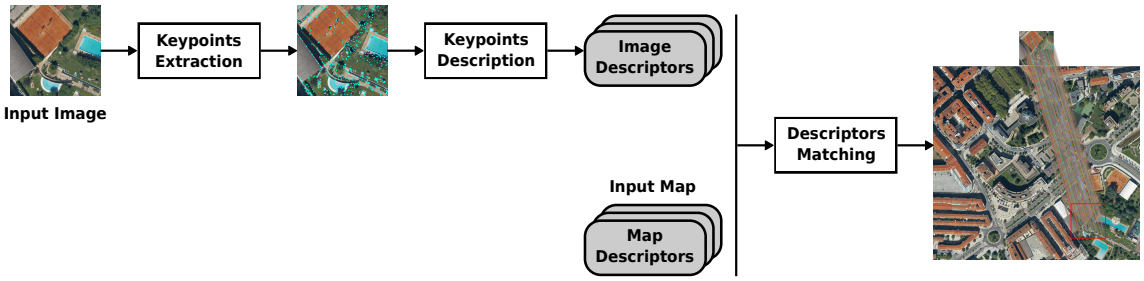


Figure 5-13: UAV image-to-map matching pipeline (images from the real application).

Each of the kernels has been implemented using HLS and encapsulated in an ARTICo^3 slot. As described above, the FAST algorithm has three parameters (i.e., detection threshold, local region size, and keypoints per region). A brief experimental evaluation of these configuration parameters has been conducted, determining a suitable set of values that render precise matching between images while being computationally lightweight. Table 5-6 lists the value selected per parameter.

Table 5-6: Configuration parameters of the FAST extraction algorithm.

Parameter	Description	Selected Values
Threshold	The intensity variation required between a pixel and its surrounding to qualify as keypoint	40 (0-255)
Grid dimensions	The amount of local regions considered when capping keypoints (grid structure)	4×4
Keypoints per local region	The maximum number of keypoint accepted per local region	2

The experiments of this validation campaign have been conducted in the two-cluster architecture described in Figure 5-2, deploying the mentioned application in a set of diverse scenarios (see Table 5-4).

The deployed application is graphically represented in Figure 5-14. The left side represents the video (as a set of frames) that the UAV camera captures, which is then processed frame by frame and mapped to the pre-existing map of the terrain shown on the right side. Each frame is a 768×768-size image, processed in 96×96-size tiles to extract keypoints and descriptors. Meanwhile, the map descriptors have already been computed beforehand. The matching stage is then performed between the run-time descriptors from the frame and the precomputed descriptors from the map. The figure shows the actual frame-by-frame mapping generated with the pipeline, which, as highlighted in the red path, enables the geolocalization of the UAV during its flight.

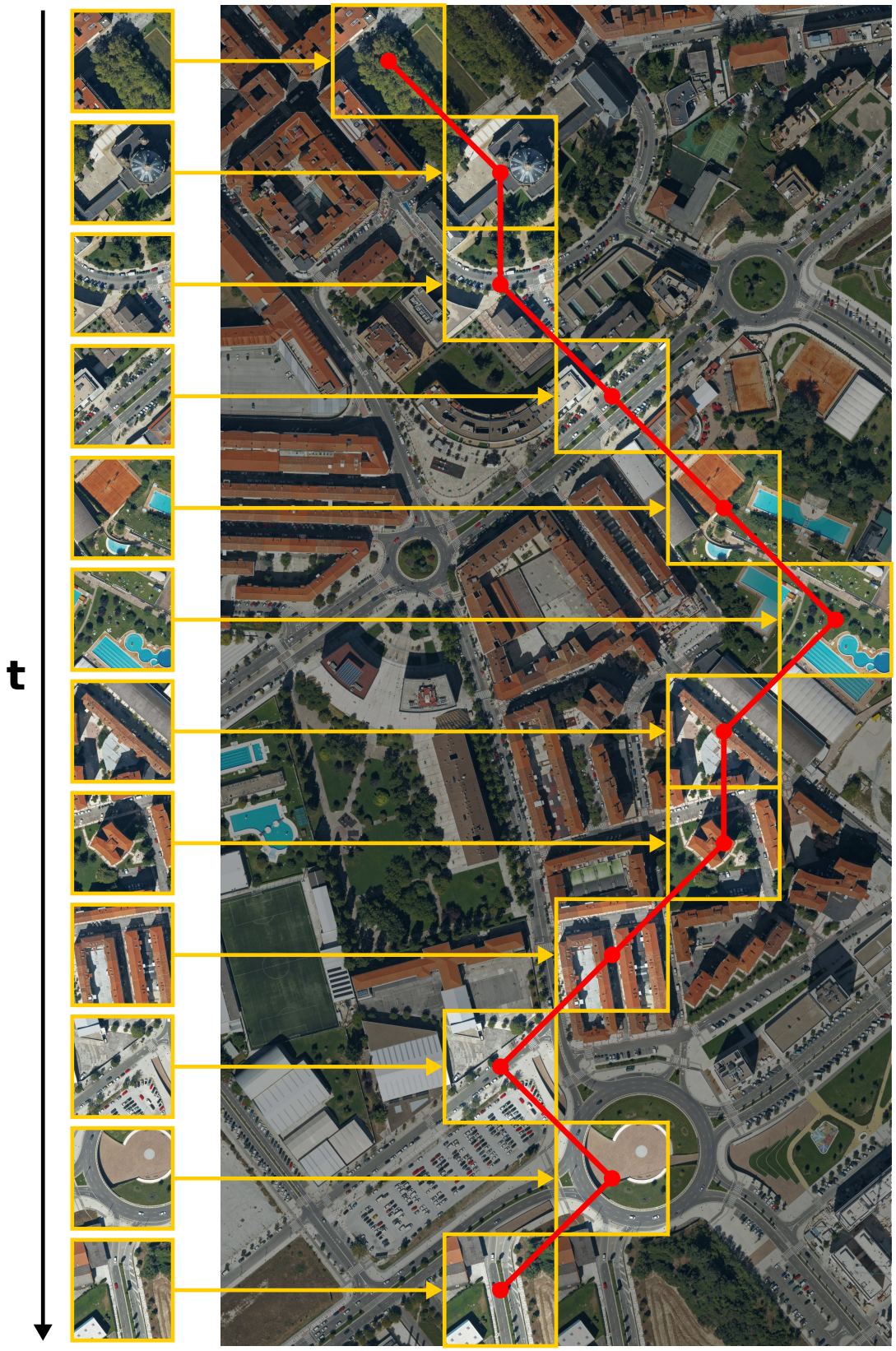


Figure 5-14: Representation of the feature extraction and matching application execution.

From this point on, the experiments are evaluated in terms of power and performance metrics extracted from the average per-frame processing of this video. The following sections explore the design space and evaluate the available trade-offs by comparing this application deployment in the different scenarios (see Table 5-4).

Single Board Results and Discussion

To exploit the diversity of each available platform, a different implementation of each feature extraction and matching kernel (i.e., FAST, BRIEF and Brute-Force matching) has been developed. Each implementation leverages the resources available per slot in each platform to deploy a higher level of internal parallelism (e.g., loop unrolling, pipelining). Moreover, since each board presents a different number of available slots, data-level parallelism has been applied by deploying several replicas of each kernel to process more data in parallel. The resource utilization of each implementation is listed, for each platform, in Table 5-7, together with the latency per kernel and the selected number of replicas. Please note that the selected number of replicas is based on previous experimental evaluation, to find the best performance per board. However, other configurations could be implemented based on application-specific needs or if resources need to be shared between applications or tenants.[4] Moreover, the modeling and scheduling components of this Thesis could also be leveraged to make run-time decisions on how many replicas of each kernel and when to place them.

Table 5-7: Resource utilization, per-kernel latency, and selected amount of per-kernel replicas for the feature extraction and matching pipeline across platforms.

Kernel Name	FAST			BRIEF			BF Matcher		
Board Type	Edge	Fog	Cloud	Edge	Fog	Cloud	Edge	Fog	Cloud
LUTs	3,801	8,212	30,094	990	3,582	4,385	1,475	4,066	14,157
FFs	2,334	2,729	10,711	795	2,017	2,504	2,209	5,028	8,618
BRAMs	16	16	46	16.5	49	41	16	20	24
DSPs	5	5	46	2	3	27	0	0	0
Latency (ms)[1]	3.628	1.367	0.954	3.508	1.204	0.850	16.166	0.998	0.435
# Replicas	1	2	6	1	2	6	2	4	4

[1] The clock frequencies per board when extracting these latency metrics are listed in Table 5-7

Figure 5-15 presents the Frames per Second (FPS) achieved on a single board in each cluster layer. The results show the FPS achieved when using one replica per kernel of the pipeline (i.e., baseline), as well as when implementing the data-level parallel configuration depicted in Table 5-7.

The results report, as one would expect, an increase in performance as data processing is migrated towards the cloud. While these results are coherent with those of the first use case, it is important to mention that in that case, the speedup was

[4] For this use case, the boards are assumed to be fully available, leveraging all the available slots.

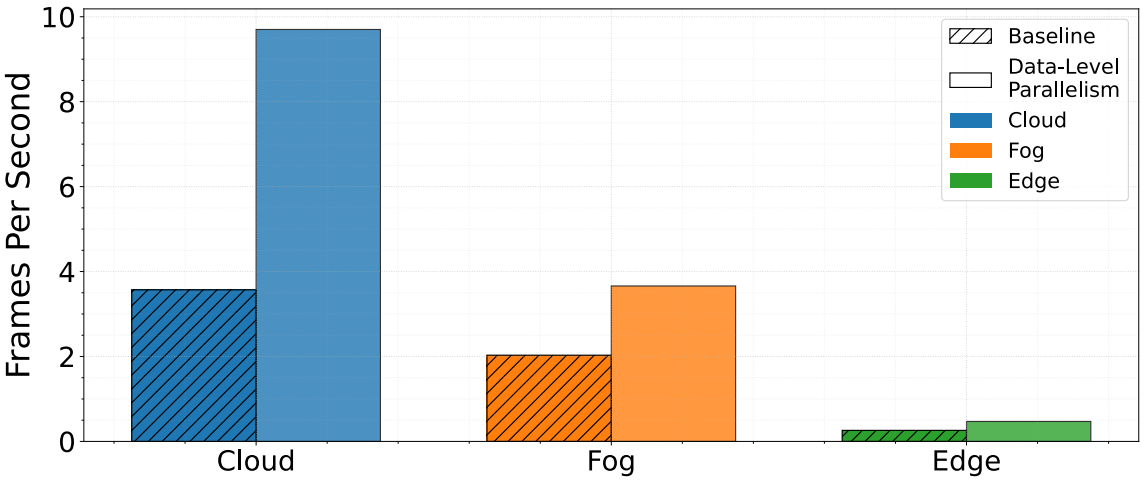


Figure 5-15: FPS achieved per (larger is better).

mainly due to the use of multiple accelerators per kernel (the kernels were identical in each board). In turn, the distinct kernel implementation per board of this use case, exploiting the variation in resources, has rendered a significant speedup even in the baseline version without data-level parallelism. This is even more prominent when data-level parallelism is leveraged, with the figure reporting even more significant speedups. Therefore, implementing kernel designs tailored to each layer enables the exploration of a broader range of solutions in terms of performance, and complements the intrinsic data-level parallelism of the multiple reconfigurable slots (whose number also changes depending on the specific layers).

Please note that power consumption is not reported since the results are equivalent to those of the first use case (Figure 5-4). However, energy efficiency will be discussed in a subsequent section.

Layer-Level Results and Discussion

Sometimes, a single-board deployment might not satisfy specific application requirements. In such scenarios, the use of cloud-edge continuum excels, enabling multi-board deployments that can leverage collaborative executions to achieve higher performance standards.

For instance, in this particular use case, depending on the UAV's speed or the target time resolution, the FPS goal might vary significantly. An interesting approach would be to share the video processing between multiple boards to collaborate on a unique output. A way to exploit such a strategy could be the use of a swarm of UAVs, where each of them features a set of nodes (typically edge and fog nodes) that, combined with the proposed cloud-edge continuum infrastructure, can work together to speed up a given processing load.

This strategy has been evaluated by distributing video processing across multiple boards at each layer (edge and fog nodes could represent swarms of UAVs). Each

board receives the same UAV point-of-view image and processes it at run time. The image is then matched to the map, but each board works against a different partition of the precomputed map. Finally, the per-board results are combined to produce the global image-to-map match. This parallelization approach is preferred over deploying different, consecutive UAV images in each node and forcing it to match the whole map: doing so would increase throughput (the full pipeline is replicated) but would not reduce the latency of a single match. In contrast, partitioning the map reduces the data each board must process, thereby lowering per-frame latency, a critical advantage for real-time applications.

Figure 5-16 presents the FPS achieved by scaling this strategy among the same layer. Please note that in the cloud and fog layers, one cluster features just one board, while at the edge, a cluster contains four nodes.

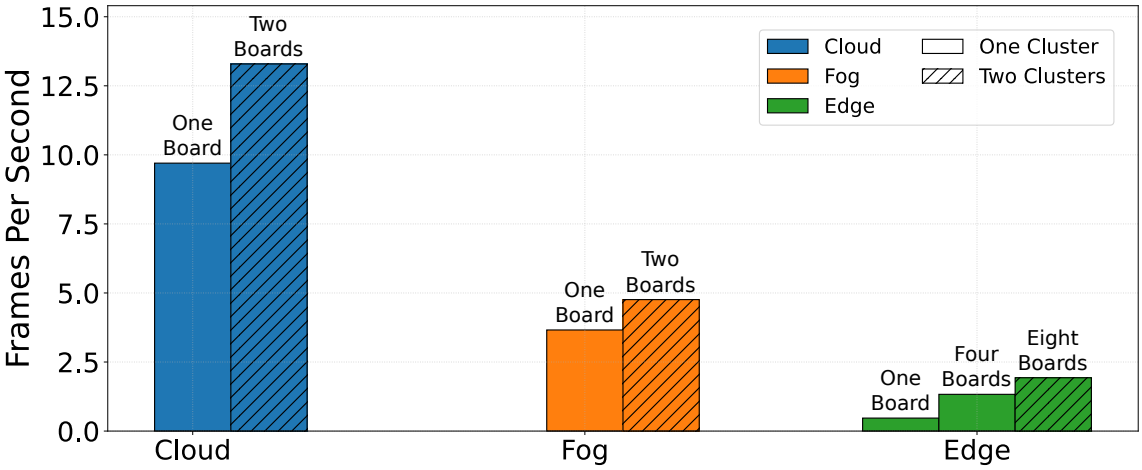


Figure 5-16: FPS achieved per layer and nodes (larger is better).

As in the first use case, performance improves as more nodes are assigned to the workload. However, since those nodes accelerate only part of the pipeline, the overall speedup is bounded by the time spent in the non-parallelized section. Consequently, the gains are more moderate than in the first use case, where adding nodes parallelizes the entire application end-to-end.

Again, the power consumption is equivalent to that reported in the first use case.

Continuum-Level Results and Discussion

For the trade-off analysis in this cloud-edge continuum, Figure 5-17 shows a scatter plot of the FPS and FPS per Watt (FPW) achieved when running the feature extraction and matching pipeline under each deployment scenario listed in Table 5-4. For this use case, entire-cluster scenarios are excluded from the comparison.[5] The horizontal

[5]As the non-parallel section of the pipeline limits acceleration and runs significantly slower on lower-end boards, multi-layer setups are not beneficial here: faster boards are constrained by the slower ones.

axis reports the achieved FPS per scenario. The vertical axis shows FPW, used as an energy-efficiency metric and normalized to the least energy-efficient scenario (i.e., the two-cluster cloud layer). The size of the bubble and the number in its center indicate the number of nodes in the scenario. The bubble outline represents the cluster scope: thin edges for single-cluster scenarios and thick edges for two-cluster scenarios.

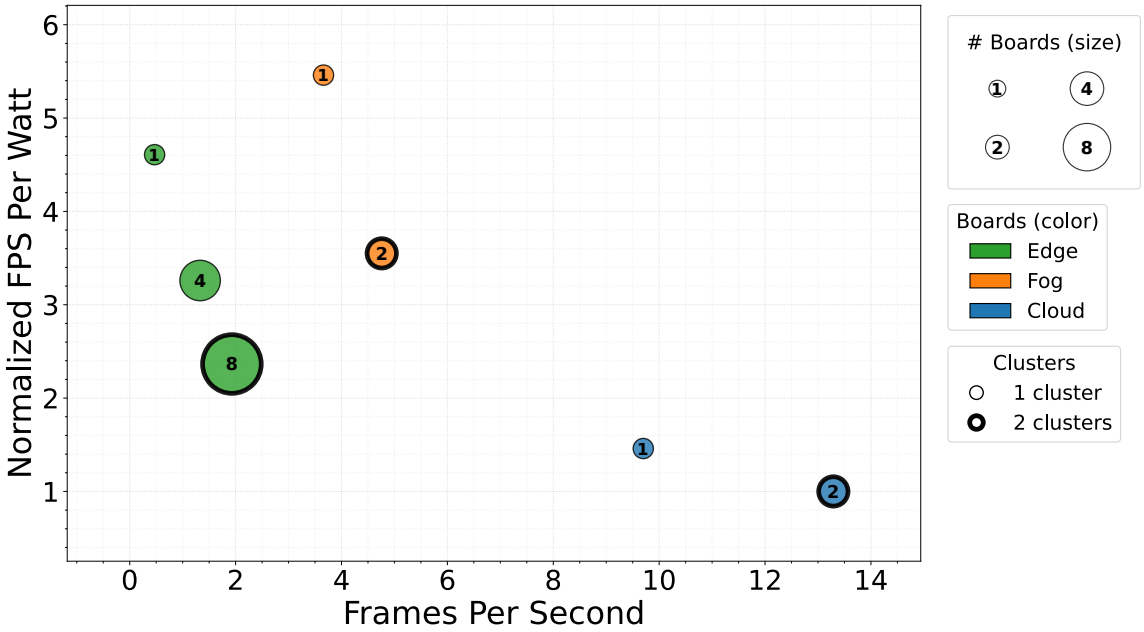


Figure 5-17: FPS against FPW per layer and nodes, normalized to the worst scenario in FPW (larger is better in both axes). Bubble size indicates the number of boards. Color indicates board type. Thick edge means two-cluster scenario; thin edge, one-cluster scenario.

With regard to performance, the results are evident. By exploiting resource abundance in the cloud through higher intra-kernel parallelism, FPS are maximized. Moving towards the edge gradually reduces the performance. This phenomenon intensifies when scaling the number of nodes, resulting in increased performance compared to their single-board counterparts. A clear progression of operating points, performance-wise, can be explored by altering the type of boards and the number of nodes for the deployment.

Talking about energy efficiency, some conclusions can be drawn. First, it is observed that scaling the number of nodes (for the same layer) reduces energy efficiency. The reason is that, as observed in the previous section, the application speedup does not scale linearly with the number of nodes, while power consumption does; therefore, energy efficiency decreases as the deployment is scaled up. Second, the type of board plays a critical role. Cloud boards, with their massive power drain, become the least energy-efficient solution. In turn, between the fog and the edge, the battle is less evident: the power consumption between the edge and cloud alternatives is not so far apart. However, the pipeline implementation in the fog boards is considerably more performing, hence, energy efficiency in the fog stays ahead.

It is interesting to observe that, contrary to the first use case where edge layers were the clear winners in energy efficiency, in this case, those deployments seem not to be the best option in any case, with always at least another option more performing or efficient. However, there are other trade-offs to consider, such as the degree of privacy or data transfer latency, where the edge would always be the first option.

As with the first use case, many more configurations can be implemented by adjusting knobs such as the number of nodes or intra-kernel parallelism. The entire graph space could be covered by a specific cluster configuration, enabling a broad amount of trade-offs and operating points, but such a solution would only be viable with the resource management methodologies proposed in this Thesis.

5.4.3 Overall Results

Across both use cases, performance increases as computation moves from edge to fog to cloud, with single-board results reflecting the larger resource availability at higher layers. This progression is amplified in the second use case, where per-platform kernel specializations deliver substantial improvements even before adding replicas.

Scaling nodes within a layer increases performance in both studies, but to different extents. In the benchmark-based use case, aggregating many identical edge nodes narrows the gap with the cloud; for example, the two-cluster edge layer (16 nodes) approaches the two-cluster cloud layer despite the higher-end devices of the latter. In the feature extraction and matching use case (single application pipeline), speedups are more moderate because the non-parallel section bounds acceleration; the resulting sublinear scaling explains why additional nodes increase FPS less than proportionally.

Energy–efficiency trade-offs differ between use cases. In the first use case, edge deployments are the most energy efficient since, when nodes are scaled, speedups can compensate for the higher power so that the total energy remains competitive. In the second use case, energy efficiency (FPW) generally decreases as nodes are added within a layer because speedup grows sublinearly while power grows roughly proportionally; cloud configurations are least efficient, and fog often provides the best balance between FPS and FPW.

Heterogeneous (mixed-board) deployments are beneficial when the workload can be balanced across nodes, as observed in the entire-cluster scenarios of the first use case. In contrast, the second use case excludes those mixed scenarios because the pipeline's non-parallel section runs much slower on lower-end boards, causing faster boards to be limited by slower ones and yielding little net benefit.

Overall, the results show a broad landscape of operating points reaching performance, energy efficiency, latency and privacy, ruled by layer choice, node count and kernel design. Importantly, the proposed resource management methodology enables exploring these diverse points (across layers, with or without specialization, and at various scales) so that different application natures can exploit the most suitable trade-offs within the cloud–edge continuum.

5.5 Conclusions

This chapter has presented the end-to-end integration of this Thesis components (i.e., the workload management infrastructure, the run-time workload characterization, and the workload optimization methodology) within a multi-cluster cloud-edge continuum architecture. The resulting system deploys dynamic workloads across heterogeneous FPGA-based nodes, monitors them in real time, and optimizes their execution locally and continuously. Validation has been carried out on a two-cluster Kubernetes architecture federated with Liqo, using a benchmark-driven synthetic workload to stress the limits of portability, scalability, and optimization.

The validation campaigns conducted in this chapter highlight two main contributions of the integrated system and demonstrate its applicability to dynamic and distributed environments.

First, the integration offers an autonomous, portable, and optimized workload execution stack that transforms heterogeneous FPGA nodes into self-managing resources. Containerized accelerators and platform-agnostic deployment remove target-specific requirements. Run-time monitoring and characterization provide local insight into performance and power consumption. The lightweight metaheuristic optimization applies that insight to select tasks and replicas while mitigating kernel interaction. In combination, these capabilities reduce manual intervention, improve reproducibility, and maintain low operational overheads.

Second, the two-cluster federation demonstrates that the integrated approach is scalable and generalizable across the cloud-edge continuum. The same abstractions and components operate equally from edge to cloud, allowing computation to be moved or replicated across layers and clusters. For the case of these proposed use case scenarios, a clear set of trade-offs has been identified: performance improves when getting closer to cloud-grade nodes or scaling node counts within a layer, while energy efficiency generally increases as execution moves toward the edge. Mixed-layer (i.e., entire-cluster) deployments represent intermediate options that can be tuned to specific constraints. Importantly, the integrated system makes this design space explorable at run time, enabling the choice of the operating point that best fits the workload's requirements.

Together, these findings validate the integration of infrastructure, incremental modeling, and conflict-aware scheduling, which serve as a practical foundation for distributed FPGA-based computing across the cloud-edge continuum. By unifying the proposed components and mechanisms in a single architecture, this chapter demonstrates a path to scalable and resource-aware acceleration that remains portable across devices and adaptable to the application's needs.

Original contribution 5-1 *A multi-cluster cloud-edge continuum architecture that integrates workload management, characterization and optimization, enabling the exploration of a vast amount of operating points and trade-offs.*

Chapter

6 CONCLUSIONS, IMPACT AND FUTURE LINES OF WORK

This chapter presents the conclusions extracted from the research conducted in this Thesis. A summary of the main contributions of the developed work is reported next. The obtained results are then evaluated in terms of dissemination. The chapter concludes by discussing the future lines of research that open up beyond this Thesis.

6.1 Conclusions of the Thesis

Reconfigurable multi-accelerator systems based on FPGAs combine the performance and energy efficiency of hardware acceleration with the flexibility required in dynamic and heterogeneous computing environments. However, their effective integration into the cloud-edge continuum faces challenges in terms of abstraction, portability, and run-time adaptability. This Thesis has addressed these challenges by making contributions that span multiple levels: infrastructure, monitoring, workload characterization, and adaptive scheduling, enabling the transparent and efficient deployment of dynamic hardware-acceleration workloads across the diverse FPGA-based platforms of the cloud-edge continuum.

Workload Management Infrastructure for the Cloud-Edge Continuum

An infrastructure has been designed to support the deployment, execution, and management of dynamic workloads on reconfigurable multi-accelerator systems in a hardware-agnostic manner. The solution extends the ARTICo^3 framework to provide the mechanisms needed to support cloud-based FPGAs and multi-tenant execution.

To enable their transparent deployment across heterogeneous nodes, workloads are virtualized in containers and orchestrated via Kubernetes, supporting multi-cluster environments through Liqo. This combination allows for seamless workload deployment between low-end edge boards and high-performance cloud-grade FPGA nodes without user intervention or knowledge of hardware specifics.

The infrastructure includes a run-time monitoring framework capable of capturing high-resolution and lightweight power/performance traces, depending on system constraints. Its adaptability grants non-intrusive monitoring across diverse platforms.

Validation results confirm that the infrastructure scales effectively across heterogeneous platforms, introduces minimal run-time overhead, and provides a wide range of trade-offs between performance, energy consumption, and resource availability. These findings demonstrate the practicality of deploying reconfigurable devices in the cloud-edge continuum while preserving their performance and efficiency advantages.

Run-Time Workload Characterization

A methodology for the run-time characterization of dynamic workloads in reconfigurable multi-accelerator systems has been developed. It captures and models the significant impact of kernel interactions on workload performance, which has been proven to cause execution time variations of up to 500%.

This Thesis initially presented an offline characterization stage to evaluate the feasibility of kernel interaction estimation, which has then been refined for run-time characterization through the use of incremental ML-based models. A dedicated orchestration mechanism coordinates the learning process to minimize induced computing overhead, ensuring that model updates remain efficient and responsive to dynamic run-time workload changes and environmental conditions.

Experiments show that the methodology achieves prediction accuracy within 4% of the continuous learning alternative while reducing the induced overhead from 20.91% to 4.49%. In dedicated tests, it predicts complex kernel interaction effects with a mean relative error below 0.6%. The method has been validated on platforms with different architectures, reconfiguration mechanisms, and power profiles, maintaining accuracy and low overhead without platform-specific tuning. This robustness demonstrates that it can be applied across the full range of platforms in the cloud-edge continuum.

Adaptive Run-Time Workload Optimization

Building on the run-time monitoring and characterization capabilities, a metaheuristic-based scheduling methodology has been developed to optimize workload execution in reconfigurable multi-accelerator systems at run time. The conflict-aware scheduling strategy, implemented using the CSA algorithm, leverages run-time workload characterization to make informed decisions that adapt to changing system conditions and task interactions.

The scheduler supports multi-objective optimization, allowing for the prioritization of either makespan reduction or fair resource allocation. The validation campaign demonstrates that adaptive scheduling significantly reduces workload execution times (by up to 11% in the worst case), even when accounting for modeling and optimization overhead, leading directly to energy savings that are essential for resource-constrained edge environments. When configured for fairness, it reduces task waiting times, offering flexibility to meet different user requirements. Moreover, a sensitivity analysis of the scheduler parameters has been conducted, providing insight into how parameter tuning affects optimization results.

The proposed workload optimization methodology has been successfully deployed on FPGAs ranging from lower-end devices to higher-end alternatives, maintaining performance and energy benefits without requiring architecture-specific adjustments.

End-to-End Components Integration

An end-to-end integration of the workload management infrastructure, run-time characterization and conflict-aware optimization has been carried out into a two-cluster cloud-edge continuum, turning heterogeneous FPGA nodes into self-management resources. Beyond the inherent portability, the unified stack (containerized accelerators on Kubernetes/Liqo, real-time monitoring/modeling and lightweight metaheuristic scheduling) makes the exploration of the vast design space of the continuum practical at run time. The computation can be placed, replicated, or partitioned across layers and clusters to realize Pareto operating points spanning performance, energy, latency and privacy, without platform-specific changes and with low overhead. In short, the integration enables both the exploration and the exploitation of a broad landscape of continuum trade-offs, providing a practical foundation for resource-aware acceleration that adapts to diverse application needs.

Final Remarks

The validation results confirm that the integration of the workload management, run-time monitoring and characterization, and the metaheuristic-based scheduling produces a practical framework for managing reconfigurable multi-accelerator systems in heterogeneous and dynamic environments. The proposed methodology scales across a wide range of platforms, maintains portability without architecture-specific tuning, and yields consistent performance when adapting to variations in workloads and system conditions. Beyond demonstrating its practical applicability to edge, fog, and cloud scenarios, this research provides a structured methodology for combining infrastructure, modeling, and scheduling in a unified design. This methodological contribution supports reproducibility, facilitates extension to other contexts, and advances the understanding of how reconfigurable systems can be efficiently managed in distributed cloud-edge continuum environments.

However, this Thesis also has some limitations, and several features expected in a production-grade system remain outside its scope. For example, encapsulating not only the workload tasks but also each component of this Thesis as a container and leveraging certain Kubernetes mechanisms (i.e., *DaemonSet*) would allow for the automatic deployment of the components on every node, removing the need for manual startup. Likewise, integrating advanced fault-tolerance mechanisms could improve system robustness, and fully automated bitstream generation and distribution services could enhance user experience. In this Thesis, these aspects are managed manually, as the focus is on validating the proposed methodologies rather than delivering a finalized production deployment.

6.2 Summary of Main Contributions

This Thesis presents the work conducted to enable resource management of reconfigurable multi-accelerator systems within the context of the cloud-edge continuum. The main contributions of this development, highlighted during the manuscript, are summarized in this section according to the goals described in Section 1.2.

- **[Infrastructure]** An open-source platform-agnostic infrastructure to deploy and monitor dynamic workloads on reconfigurable multi-accelerator systems in the cloud-edge continuum.
- **[Infrastructure]** The extension of the ARTICo^3 framework for cloud-based FPGAs and multi-tenant execution.
- **[Infrastructure]** A virtualization methodology to support the seamless deployment of hardware accelerators across the cloud-edge continuum.
- **[Monitoring]** An open-source, composable monitoring framework for synchronized power consumption and performance traces acquisition on reconfigurable multi-accelerator systems.
- **[Kernel Interaction Analysis]** An evaluation of the impact on power consumption and performance of the interaction between kernels when executing dynamic workloads in reconfigurable multi-accelerator systems.
- **[Modeling]** A device-agnostic, data-driven methodology to predict power consumption and performance in reconfigurable multi-accelerator systems.
- **[Modeling]** An incremental data-driven modeling extension to predict power consumption and computing performance in reconfigurable multi-accelerator systems at run time and on demand.
- **[Modeling]** A resource-aware model orchestration mechanism to perform run-time adaptive incremental learning and keep models updated while minimizing system overhead.
- **[Scheduling]** A metaheuristic-based strategy that relies on data-driven predictive models to schedule dynamic workloads in reconfigurable multi-accelerator systems, optimizing and trading off energy and performance on the one hand, and fair use of resources on the other hand, at run time.
- **[Scheduling]** A scheduling policy that uses run-time kernel interaction estimations to guide scheduling decisions in reconfigurable multi-accelerator systems.
- **[Cloud-Edge Continuum]** A methodology to grant user-agnostic compatibility across the diverse set of hardware nodes present in the cloud-edge continuum.

- **[Cloud-Edge Continuum]** A multi-cluster cloud-edge continuum architecture that integrates workload management, characterization and optimization, enabling the exploration of a vast amount of operating points and trade-offs.
- **[Validation]** A dwarf-based characterization and validation strategy for the proposed technology based on HLS benchmarks.

6.3 Impact of the Thesis

This section analyzes the impact of this Thesis by quantitatively reporting the obtained results. The results are categorized into dissemination activities directly related to this Thesis, research projects where the work of this Thesis has been either developed or applied, M. Sc. theses co-supervised that are associated with this Thesis, and collaborations with other research groups.

The results report is presented in the form of categorized lists. Please note that for each list entry, the collaboration effort with other research groups is highlighted, its relationship with this Thesis is described, and references to the specific sections of this document where the content is presented are provided.

6.3.1 Publications and Dissemination

This section lists every dissemination activity directly related to the work of this Thesis and describes this relationship.

Journal Publications

[Encinas'25b] J. Encinas, A. Rodríguez, A. Otero, "Leveraging Incremental Machine Learning for Reconfigurable Systems Modeling under Dynamic Workloads", in *ACM Trans. Reconfigurable Technol. Syst.*, vol. 18, no. 1, Mar 2025, ISSN 1936-7406, doi:10.1145/3715154

2023 JCR impact factor: 3.1 (Q2)

This publication presents and evaluates the incremental extension of the modeling methodology described in Section 3.6, including the extension of the workload management infrastructure and an assessment of its portability.

[Encinas'24] J. Encinas, A. Rodríguez, A. Otero, E. de la Torre, "Data-driven modeling of reconfigurable multi-accelerator systems under dynamic workloads", in *Microprocessors and Microsystems*, vol. 107, p. 105 050, 2024, ISSN 0141-9331, doi: 10.1016/j.micpro.2024.105050

2023 JCR impact factor: 1.9 (Q2)

This publication presents and evaluates the offline implementation of the data-driven modeling methodology to characterize workload in reconfigurable multi-accelerator systems described in Section 3.5, together with the baseline workload management infrastructure described in Chapter 2 and the kernel interaction analysis described in Section 3.3.

Conference Publications

[Encinas'25a] J. Encinas, F. Ratto, I. Díez de Ulzurrun, C. Rubattu, A. Rodríguez, F. Palumbo, A. Otero, "Modular and Composable Framework for Multipurpose Monitoring in Heterogeneous FPGA-Based Systems", in *Embedded Computer Systems: Architectures, Modeling, and Simulation*, Springer Nature Switzerland, Cham, in press, 2025, ISBN 978-3-031-78380-7

Joint effort with Università degli Studi di Cagliari and Università degli Studi di Sassari

This publication presents the monitoring framework described in Section 2.3.3 and covers an extensive evaluation, assessing its portability and composability, as described in Section 2.5.2.

[Díez De Ulzurrun'24b] I. Díez De Ulzurrun, J. Encinas, A. Rodríguez, A. Otero, "Cloud-Edge Continuum Infrastructure for Reconfigurable Multi-Accelerator Systems", in *2024 39th Conference on Design of Circuits and Integrated Systems (DCIS)*, pp. 1–6, 2024, doi:10.1109/DCIS62603.2024.10769198

This publication presents and evaluates the extension of workload management infrastructure to the cloud-edge continuum scenario described in Section 2.4. It covers the ARTICo^3 extension to support cloud-based FPGA and the hardware acceleration virtualization.

[Encinas'21] J. Encinas, A. Rodríguez, A. Otero, E. De La Torre, "Run-Time Monitoring and ML-Based Modeling in Reconfigurable Multi-Accelerator Systems", in *2021 36th Conference on Design of Circuits and Integrated Systems (DCIS)*, pp. 1–7, 2021, doi:10.1109/DCIS53048.2021.9666187

This publication presents a first assessment on using ML-based models for predicting power consumption and performance in reconfigurable multi-accelerator systems. The insights from this work were the seeds for the development of the workload management infrastructure described in Chapter 2 and the refinement of the modeling methodology described in Chapter 3.

Other Dissemination Activities

The intermediate results of this Thesis have been presented at international PhD forums, producing three separate poster publications.

[Encinas'22] J. Encinas, "ML-Based Modeling and Virtualization of Reconfigurable Multi-Accelerator Systems.", in *CPS Summer School, PhD Workshop*, 2022, https://ceur-ws.org/Vol-3252/short2.pdf

[Encinas'23] J. Encinas, "Run-Time ML-Based Modeling and Management of Reconfigurable Multi-Accelerator Systems with Virtualization Support", in *2023 38th Conference on Design of Circuits and Integrated Systems (DCIS)*, pp. 1–2, 2023

[Encinas'25c] J. Encinas, A. Rodríguez, A. Otero, "ML-Based Resource Management of Reconfigurable Systems in the Cloud-Edge Continuum", in *2025 Design, Automation & Test in Europe Conference (DATE)*, pp. 1–2, 2025

The components presented in this Thesis have been used as the core of a tutorial on resource management in the cloud-edge continuum.

Tutorial – "Resource Orchestration in the Cloud-Edge Continuum", in *CPS Summer School 2025: Designing Cyber-Physical Systems - From concepts to implementation*, Alghero, Italy, September 2025.

Aligned with the research context of this Thesis, a hackathon-style hands-on lab has been co-organized at two editions of the Cyber-Physical Systems Summer School.

Hackathon (co-organizer) – "Creative Lab", in *CPS Summer School Sardinia 2024*, Sardinia, Italy, September 2024.

Hackathon (co-organizer) – "Creative Lab", in *CPS Summer School Sardinia 2024*, Sardinia, Italy, September 2025.

6.3.2 Research Projects

Some of the results of this Thesis are, to some extent, derived from or applied to several research projects. The list of projects where the author of this Thesis has been actively involved is provided in this section.

KYKLOS 4.0 – An Advanced Circular and Agile Manufacturing Ecosystem based on Rapid Reconfigurable Manufacturing Process and Individualized Consumer Preferences

H2020-DT-2019-1 (872570)

The initial version of the modeling methodology proposed in this Thesis was developed and evaluated as part of one of the project's use cases. The implementation of ML-based modeling at the edge was applied to detect and track objects for collision-avoidance purposes. Such implementation was the seed for the lightweight modeling approach presented in this Thesis.

Almacenamiento Inteligente basado en Baterías de Segunda Vida para Integración de Renovable

Spanish R&D National Program (TED2021-132768B-I00)

The modeling methodology of this Thesis was refined as part of this project, where ML-based models were used to characterize the impedance of a three-phase network load at the edge, guiding the control of a power converter.

MYRTUS – Multi-Layer 360° Dynamic Orchestration and Interoperable Design Environment for Compute-Continuum Systems

HORIZON-CL4-2023-DATA-01 (101135183)

In this project, the main components of this Thesis were developed, refined and evaluated. The integration of infrastructure components, the incremental extension of the modeling methodology, and the development of the scheduling algorithm were key components of this project. The evaluation of each element was conducted first using synthetic benchmarks and then as part of one of the project's use cases. This project will facilitate the development of part of the future work outlined in this Thesis.

6.3.3 Co-Supervised Works

This section reports the M. Sc. theses co-supervised by the author of this Thesis, which are directly related to the Thesis itself by either supporting the development of its components or evaluating alternative approaches.

[Díez De Ulzurrun'24a] I. Díez De Ulzurrun, "Virtualization of PCIe-Based Reconfigurable Multi-Accelerator Systems for the Edge-Cloud Continuum", M.Sc. thesis, Universidad Politécnica de Madrid, Jun 2024, https://oa.upm.es/84007

[Fernández'25] E. J. Fernández, "Federated Learning for Modeling of Reconfigurable Multi-Accelerator Systems in the Cloud-Edge Continuum", M.Sc. thesis, Universidad Politécnica de Madrid, 2025

6.3.4 Collaborations

This section summarizes some of the activities of this Thesis that result from the collaboration with other researchers. Please note that some have already been described in the corresponding publications.

The work on the composable monitoring framework and the infrastructure extension to multi-cluster topologies was partly carried out during a 3-month research stay at *Università degli Studi di Cagliari*, in collaboration with *Università degli Studi di Sassari*, under the supervision of Professor Francesca Palumbo. In particular, the evaluation of the monitoring framework and the integration of the workload management infrastructure with Liqo were developments performed together with Francesco Ratto, Claudio Rubattu and Francesca Palumbo, with support from Luca Castello of ArubaKube, a MYRTUS partner.

All the activity related to hardware acceleration virtualization, either at the node level or the continuum level, has been developed together with Iñigo Díez de Ulzurrun (another researcher from the author's group) and is included in his M. Sc thesis.

The baremetal support of the monitoring framework for the RISC-V architecture has been implemented in collaboration with Jorge Brihuega (another researcher from the authors' group).

6.4 Future Lines of Work

The research work presented in this Thesis sets the foundation for a unified methodology that integrates FPGA-based acceleration into the cloud-edge continuum, combining platform-agnostic deployment, run-time workload characterization, and adaptive scheduling. The proposed infrastructure, modeling approach, and scheduling strategy have been validated across heterogeneous devices and operational contexts, demonstrating portability, scalability, and efficiency. Nonetheless, several promising research directions emerge from the limitations and assumptions of the current design, which could further expand its impact on the field.

Infrastructure Improvements

The current implementation focuses on deploying and managing independent workloads or loosely coupled tasks in reconfigurable multi-accelerator systems. While this approach is effective for a broad range of applications, many real-time and streaming workloads exhibit complex data dependencies between tasks, requiring execution models that go beyond monolithic workflows. Future extensions could incorporate dynamic dataflow graphs in which accelerators exchange intermediate results directly, without returning to main memory, thus reducing latency and improving throughput. This would involve the introduction of new data movement patterns, such as direct slot-to-slot transfers, as well as deadline-aware deployment mechanisms capable of adapting to evolving dependencies at run time. Such capabilities would significantly expand the applicability of the infrastructure to newer domains.

Modeling Improvements

The incremental learning methodology proposed in this Thesis has proven highly accurate in predicting execution time and power consumption under kernel interaction effects, with low overhead and strong portability. However, two research lines appear particularly promising for further development. The first is the introduction of federated learning techniques, enabling geographically distributed FPGA nodes to collaboratively train global performance and power models without exchanging raw monitoring data. This approach would preserve privacy, reduce communication costs, and accelerate model adaptation to new devices or workloads by leveraging collective knowledge from across the continuum. Moreover, since each node may execute a different workload and consequently train a node-specific model that is not transferred when tasks are migrated to another node, federated learning could complement workload mobility by maintaining a global model that generalizes across nodes and remains useful as workloads migrate. The second is the integration of time-series-based prediction models, capable of forecasting workload evolution over time, which allows the system to estimate upcoming changes in workload composition, contention levels, or resource availability. Such predictive capabilities would enable the scheduling methodology to take proactive rather than reactive decisions, improving stability and responsiveness under dynamically changing conditions.

Scheduling Improvements

The conflict-aware scheduler presented in this Thesis operates at the node-level scale, optimizing task-to-slot allocations within a single FPGA. Building on this principle, a higher-level scheduler could be developed to operate across the entire cloud-edge continuum, partitioning workloads among multiple FPGA nodes while accounting for network latency, energy budgets, and execution deadlines. This scheduler could exploit information from local predictive models, global models obtained through federated learning, and time-series forecasts of workload evolution to make higher-level decisions that optimize both local and global objectives. Such a hierarchical approach, in which global scheduling determines the coarse-grain distribution of tasks and local scheduling refines the allocation at each local device, could enable more efficient resource usage, improve predictability, and further enhance the integration of FPGAs into large-scale distributed computing environments.

Bibliography

[Adhi'18] A. Adhi, B. Santosa, N. Siswanto, "A meta-heuristic method for solving scheduling problem: crow search algorithm", in *IOP Conference Series: Materials Science and Engineering*, vol. 337, no. 1, p. 012 003, apr 2018, doi:10.1088/1757-899X/337/1/012003. [17, 137, 144]

[Agne'14] A. Agne, M. Happe, A. Keller, E. Lübbers, B. Plattner, M. Platzner, C. Plessl, "ReconOS: An Operating System Approach for Reconfigurable Computing", in *IEEE Micro*, vol. 34, no. 1, pp. 60–71, 2014, doi:10.1109/MM.2013.110. [27, 29]

[Akgün'20] G. Akgün, L. Kalms, D. Göhringer, "Resource efficient dynamic voltage and frequency scaling on Xilinx FPGAs", in *Applied Reconfigurable Computing. Architectures, Tools, and Applications: 16th International Symposium, ARC 2020, Toledo, Spain, April 1–3, 2020, Proceedings 16*, pp. 178–192, Springer, 2020. [33]

[Akhter'06] S. Akhter, J. Roberts, *Multi-core programming*, vol. 33, Intel press Hillsboro, Oregon, 2006. [2]

[Al Qassem'20] L. M. Al Qassem, T. Stouraitis, E. Damiani, I. A. M. Elfadel, "A Remote FPGA Laboratory as a Cloud Microservice", in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, 2020, doi:10.1109/ISCAS45731.2020.9181123. [31]

[Alahi'12] A. Alahi, R. Ortiz, P. Vandergheynst, "FREAK: Fast Retina Keypoint", in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 510–517, 2012, doi: 10.1109/CVPR.2012.6247715. [189]

[Aldham'11] M. Aldham, J. Anderson, S. Brown, A. Canis, "Low-cost hardware profiling of run-time and energy in FPGA embedded processors", in *ASAP 2011 - 22nd IEEE International Conference on Application-specific Systems, Architectures and Processors*, pp. 61–68, 2011, doi:10.1109/ASAP.2011.6043237. [32]

[Alismail'23] S. Alismail, D. Koch, "Efficient Resource Scheduling for Runtime Reconfigurable Systems on FPGAs", in *2023 33rd International Conference on Field-Programmable*

Logic and Applications (FPL), pp. 123–129, 2023, doi: 10.1109/FPL60245.2023.00025. [140, 142]

[Amarís'16] M. Amarís, R. Y. de Camargo, M. Dyab, A. Goldman, D. Trystram, "A comparison of GPU execution time prediction using machine learning and analytical modeling", in *2016 IEEE 15th International Symposium on Network Computing and Applications (NCA)*, pp. 326–333, 2016, doi: 10.1109/NCA.2016.7778637. [91, 92]

[Aron'22] R. Aron, A. Abraham, "Resource scheduling methods for cloud computing environment: The role of meta-heuristics and artificial intelligence", in *Engineering Applications of Artificial Intelligence*, vol. 116, p. 105 345, 2022, ISSN 0952-1976, doi: https://doi.org/10.1016/j.engappai.2022.105345. [144]

[Asanovic'06] K. Asanovic, R. Bodik, B. Catanzaro, J. Gebis, P. Husbands, K. Keutzer, D. Patterson, W. Plishker, J. Shalf, S. W. Williams, "The landscape of parallel computing research: A view from berkeley", in , 2006. [54]

[Askarzadeh'16] A. Askarzadeh, "A novel metaheuristic method for solving constrained engineering optimization problems: Crow search algorithm", in *Computers & Structures*, vol. 169, pp. 1–12, 2016, ISSN 0045-7949, doi: https://doi.org/10.1016/j.compstruc.2016.03.001. [138, 145]

[Awad'15] M. Awad, R. Khanna, *Efficient Learning Machines: Theories, Concepts, and Applications for Engineers and System Designers*, Apress, 2015, ISBN 978-1-4302-5990-9, doi:10.1007/978-1-4302-5990-9. [97]

[Azarian'09] A. Azarian, M. Ahmadi, "Reconfigurable computing architecture survey and introduction", in *2009 2nd IEEE International Conference on Computer Science and Information Technology*, pp. 269–274, 2009, doi:10.1109/ICCSIT.2009.5234721. [12]

[Bacis'20] M. Bacis, R. Brondolin, M. D. Santambrogio, "BlastFunction: an FPGA-as-a-Service system for Accelerated Serverless Computing", in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 852–857, 2020, doi: 10.23919/DATE48585.2020.9116333. [30, 31]

[Bailey'91] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, et al., "The NAS parallel benchmarks—summary and preliminary results", in *Proceedings of*

the 1991 ACM/IEEE Conference on Supercomputing, pp. 158–165, 1991. [54]

[Baldini'14] I. Baldini, S. J. Fink, E. Altman, "Predicting GPU Performance from CPU Runs Using Machine Learning", in *2014 IEEE 26th International Symposium on Computer Architecture and High Performance Computing*, pp. 254–261, 2014, doi: 10.1109/SBAC-PAD.2014.30. [91, 92]

[Bay'06] H. Bay, T. Tuytelaars, L. Van Gool, "SURF: Speeded Up Robust Features", in A. Leonardis, H. Bischof, A. Pinz, editors, *Computer Vision – ECCV 2006*, pp. 404–417, Springer Berlin Heidelberg, Berlin, Heidelberg, 2006, ISBN 978-3-540-33833-8. [189]

[Bayram'22] F. Bayram, B. S. Ahmed, A. Kassler, "From concept drift to model degradation: An overview on performance-aware drift detectors", in *Knowledge-Based Systems*, vol. 245, p. 108 632, 2022, ISSN 0950-7051, doi: https://doi.org/10.1016/j.knosys.2022.108632, https://www.sciencedirect.com/science/article/pii/S0950705122002854. [76, 93]

[Bentaleb'22] O. Bentaleb, A. S. Belloum, A. Sebaa, A. El-Maouhab, "Containerization technologies: Taxonomies, applications and challenges", in *The Journal of Supercomputing*, vol. 78, no. 1, pp. 1144–1181, 2022, doi:10.1007/s11227-021-03914-1. [16, 39]

[Bertolino'20] M. Bertolino, R. Pacalet, L. Apvrille, A. Enrici, "Efficient Scheduling of FPGAs for Cloud Data Center Infrastructures", in *2020 23rd Euromicro Conference on Digital System Design (DSD)*, pp. 57–64, 2020, doi:10.1109/DSD51259.2020.00021. [140, 142]

[Bienia'08] C. Bienia, S. Kumar, J. P. Singh, K. Li, "The PARSEC benchmark suite: Characterization and architectural implications", in *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, pp. 72–81, 2008. [54]

[Bifet'09] A. Bifet, R. Gavaldà, "Adaptive Learning from Evolving Data Streams", in N. M. Adams, C. Robardet, A. Siebes, J.-F. Boulicaut, editors, *Advances in Intelligent Data Analysis VIII*, pp. 249–260, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, ISBN 978-3-642-03915-7. [100, 107]

[Blazewicz'83] J. Blazewicz, J. Lenstra, A. Kan, "Scheduling subject to resource constraints: classification and complexity", in *Discrete Applied Mathematics*, vol. 5, no. 1, pp. 11–24, 1983, ISSN 0166-218X, doi:https://doi.org/10.1016/0166-218X(83)90012-4. [17, 137]

[Bobda'22] C. Bobda, J. M. Mbongue, P. Chow, M. Ewais, N. Tarafdar, J. C. Vega, K. Eguro, D. Koch, S. Handagala, M. Leeser, et al., "The future of FPGA acceleration in datacenters and the cloud", in *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 15, no. 3, pp. 1–42, 2022. [3]

[Böhm'22] S. Böhm, G. Wirtz, "Cloud-edge orchestration for smart cities: A review of kubernetes-based orchestration architectures", in *EAI Endorsed Trans. Smart Cities*, vol. 6, no. 18, p. e2, 2022. [39]

[Bohr'07] M. Bohr, "A 30 Year Retrospective on Dennard's MOSFET Scaling Paper", in *IEEE Solid-State Circuits Society Newsletter*, vol. 12, no. 1, pp. 11–13, 2007, doi:10.1109/NSSC.2007.4785534. [1]

[Boutros'20] A. Boutros, E. Nurvitadhi, R. Ma, S. Gribok, Z. Zhao, J. C. Hoe, V. Betz, M. Langhammer, "Beyond Peak Performance: Comparing the Real Performance of AI-Optimized FPGAs and GPUs", in *2020 International Conference on Field-Programmable Technology (ICFPT)*, pp. 10–19, 2020, doi:10.1109/ICFPT51103.2020.00011. [2]

[Brackenbury'10] L. E. M. Brackenbury, L. A. Plana, J. Pepper, "System-on-Chip Design and Implementation", in *IEEE Transactions on Education*, vol. 53, no. 2, pp. 272–281, 2010, doi:10.1109/TE.2009.2014858. [11]

[Calonder'10] M. Calonder, V. Lepetit, C. Strecha, P. Fua, "BRIEF: Binary Robust Independent Elementary Features", in K. Daniilidis, P. Maragos, N. Paragios, editors, *Computer Vision – ECCV 2010*, pp. 778–792, Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, ISBN 978-3-642-15561-1. [189]

[Canis'13] A. Canis, J. Choi, M. Aldham, V. Zhang, A. Kammoona, T. Czajkowski, S. D. Brown, J. H. Anderson, "LegUp: An open-source high-level synthesis tool for FPGA-based processor/accelerator systems", in , vol. 13, no. 2, Sep. 2013, ISSN 1539-9087, doi:10.1145/2514740, https://doi.org/10.1145/2514740. [32]

[Carmo'17] M. S. Carmo, S. Jardim, A. V. Neto, R. Aguiar, D. Corujo, "Towards fog-based slice-defined WLAN infrastructures to cope with future 5G use cases", in *2017 IEEE 16th International Symposium on Network Computing and Applications (NCA)*, pp. 1–5, 2017, doi:10.1109/NCA.2017.8171397. [14]

[Chen'18] Z. Chen, B. Liu, *Lifelong machine learning*, Morgan & Claypool Publishers, 2018. [93]

[Compton'02] K. Compton, S. Hauck, "Reconfigurable computing: a survey of systems and software", in *ACM Comput. Surv.*, vol. 34, no. 2, p. 171–210, Jun. 2002, ISSN 0360-0300, doi:10.1145/508352.508353, https://doi.org/10.1145/508352.508353. [xxiii, 9, 11, 12, 13]

[Dennard'74] R. Dennard, F. Gaensslen, H.-N. Yu, V. Rideout, E. Bassous, A. LeBlanc, "Design of ion-implanted MOSFET's with very small physical dimensions", in *IEEE Journal of Solid-State Circuits*, vol. 9, no. 5, pp. 256–268, 1974, doi:10.1109/JSSC.1974.1050511. [1]

[DeTone'18] D. DeTone, T. Malisiewicz, A. Rabinovich, "SuperPoint: Self-Supervised Interest Point Detection and Description", in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2018. [189]

[Diessel'01] O. Diessel, H. Elgindy, "On dynamic task scheduling for FPGA-based systems", in *International Journal of Foundations of Computer Science*, vol. 12, no. 05, pp. 645–669, 2001, doi:10.1142/S0129054101000709. [17, 137]

[Dorigo'06] M. Dorigo, M. Birattari, T. Stutzle, "Ant colony optimization", in *IEEE Computational Intelligence Magazine*, vol. 1, no. 4, pp. 28–39, 2006, doi:10.1109/MCI.2006.329691. [145]

[Duhamel'22] A. Duhamel, S. Pillement, "QoS Aware Design-Time/Run-Time Manager for FPGA-Based Embedded Systems", in K. Desnos, S. Pertuz, editors, *Design and Architecture for Signal and Image Processing*, pp. 96–107, Springer International Publishing, Cham, 2022, ISBN 978-3-031-12748-9. [140, 142]

[Díez De Ulzurrun'24a] I. Díez De Ulzurrun, "Virtualization of PCIe-Based Reconfigurable Multi-Accelerator Systems for the Edge-Cloud Continuum", M.Sc. thesis, Universidad Politécnica de Madrid, Jun 2024, https://oa.upm.es/84007. [208]

[Díez De Ulzurrun'24b] I. Díez De Ulzurrun, J. Encinas, A. Rodríguez, A. Otero, "Cloud-Edge Continuum Infrastructure for Reconfigurable Multi-Accelerator Systems", in *2024 39th Conference on Design of Circuits and Integrated Systems (DCIS)*, pp. 1–6, 2024, doi: 10.1109/DCIS62603.2024.10769198. [206]

[EEMBC] "Embedded Microprocessor Benchmark Consortium Benchmarks", https://www.eembc.org/. Accessed: April 10, 2025. [54]

[Encinas'21] J. Encinas, A. Rodríguez, A. Otero, E. De La Torre, "Run-Time Monitoring and ML-Based Modeling in Reconfigurable Multi-Accelerator Systems", in *2021 36th Conference on Design of Circuits and Integrated Systems (DCIS)*, pp. 1–7, 2021, doi: 10.1109/DCIS53048.2021.9666187. [206]

[Encinas'22] J. Encinas, "ML-Based Modeling and Virtualization of Reconfigurable Multi-Accelerator Systems.", in *CPS Summer School, PhD Workshop*, 2022, https://ceur-ws.org/Vol-3252/short2.pdf. [207]

[Encinas'23] J. Encinas, "Run-Time ML-Based Modeling and Management of Reconfigurable Multi-Accelerator Systems with Virtualization Support", in *2023 38th Conference on Design of Circuits and Integrated Systems (DCIS)*, pp. 1–2, 2023. [207]

[Encinas'24] J. Encinas, A. Rodríguez, A. Otero, E. de la Torre, "Data-driven modeling of reconfigurable multi-accelerator systems under dynamic workloads", in *Microprocessors and Microsystems*, vol. 107, p. 105 050, 2024, ISSN 0141-9331, doi: 10.1016/j.micpro.2024.105050. [206]

[Encinas'25a] J. Encinas, F. Ratto, I. Díez de Ulzurrun, C. Rubattu, A. Rodríguez, F. Palumbo, A. Otero, "Modular and Composable Framework for Multipurpose Monitoring in Heterogeneous FPGA-Based Systems", in *Embedded Computer Systems: Architectures, Modeling, and Simulation*, Springer Nature Switzerland, Cham, in press, 2025, ISBN 978-3-031-78380-7. [206]

[Encinas'25b] J. Encinas, A. Rodríguez, A. Otero, "Leveraging Incremental Machine Learning for Reconfigurable Systems Modeling under Dynamic Workloads", in *ACM Trans. Reconfigurable Technol. Syst.*, vol. 18, no. 1, Mar 2025, ISSN 1936-7406, doi: 10.1145/3715154. [205]

[Encinas'25c] J. Encinas, A. Rodríguez, A. Otero, "ML-Based Resource Management of Reconfigurable Systems in the Cloud-Edge

Continuum", in *2025 Design, Automation & Test in Europe Conference (DATE)*, pp. 1–2, 2025. [207]

[Fanni'18] T. Fanni, A. Rodríguez, C. Sau, L. Suriano, F. Palumbo, L. Raffo, E. de la Torre, "Multi-Grain Reconfiguration for Advanced Adaptivity in Cyber-Physical Systems", in *2018 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, pp. 1–8, 2018, doi:10.1109/RECONFIG.2018.8641705. [39]

[Fanni'19] T. Fanni, D. Madronal, C. Rubattu, C. Sau, F. Palumbo, E. Juarez, M. Pelcat, C. Sanz, L. Raffo, "Run-time Performance Monitoring of Heterogenous Hw/Sw Platforms Using PAPI", in *FSP Workshop 2019; Sixth International Workshop on FPGAs for Software Programmers*, pp. 1–10, 2019. [32, 33]

[Fernández'25] E. J. Fernández, "Federated Learning for Modeling of Reconfigurable Multi-Accelerator Systems in the Cloud-Edge Continuum", M.Sc. thesis, Universidad Politécnica de Madrid, 2025. [208]

[Fischler'81] M. A. Fischler, R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography", in , vol. 24, no. 6, p. 381–395, Jun. 1981, ISSN 0001-0782, doi:10.1145/358669.358692. [192]

[Fleming'14] K. Fleming, H.-J. Yang, M. Adler, J. Emer, "The LEAP FPGA operating system", 09 2014, doi:10.1109/FPL.2014.6927488. [26]

[Gandhare'19] S. Gandhare, B. Karthikeyan, "Survey on FPGA Architecture and Recent Applications", in *2019 International Conference on Vision Towards Emerging Trends in Communication and Networking (ViTECoN)*, pp. 1–4, 2019, doi:10.1109/ViTECoN.2019.8899550. [9]

[Gionis'99] A. Gionis, P. Indyk, R. Motwani, et al., "Similarity search in high dimensions via hashing", in *Vldb*, vol. 99, pp. 518–529, 1999. [189]

[Giraud-Carrier'00] C. Giraud-Carrier, "A note on the utility of incremental learning", in *AI Communications*, vol. 13, no. 4, pp. 215–223, 2000, doi:10.3233/EAI-2000-193. [92]

[Gkonis'23] P. Gkonis, A. Giannopoulos, P. Trakadas, X. Masip-Bruin, F. D'Andria, "A Survey on IoT-Edge-Cloud Continuum Systems: Status, Challenges, Use Cases, and Open Issues", in *Future Internet*, vol. 15, no. 12, 2023, ISSN 1999-5903, doi:10.3390/fi15120383. [29]

[Goeders'17] J. Goeders, S. J. E. Wilton, "Signal-Tracing Techniques for In-System FPGA Debugging of High-Level Synthesis Circuits", in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 1, pp. 83–96, 2017, doi: 10.1109/TCAD.2016.2565204. [32, 33]

[Gomes'18] H. M. Gomes, J. P. Barddal, L. E. B. Ferreira, A. Bifet, "Adaptive random forests for data stream regression.", in *ESANN*, 2018. [108]

[González'20] S. González, S. García, J. Del Ser, L. Rokach, F. Herrera, "A practical tutorial on bagging and boosting based ensembles for machine learning: Algorithms, software tools, performance study, practical perspectives and opportunities", in *Information Fusion*, vol. 64, pp. 205–237, 2020, ISSN 1566-2535, doi:https://doi.org/10.1016/j.inffus.2020.07.007. [98]

[Goodarzy'20] S. Goodarzy, M. Nazari, R. Han, E. Keller, E. Rozner, "Resource Management in Cloud Computing Using Machine Learning: A Survey", in *2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pp. 811–816, 2020, doi:10.1109/ICMLA51294.2020.00132. [141]

[Goswami'22] P. Goswami, D. Bhatia, "Predicting Post-Route Quality of Results Estimates for HLS Designs using Machine Learning", in *2022 23rd International Symposium on Quality Electronic Design (ISQED)*, pp. 45–50, 2022, doi: 10.1109/ISQED54688.2022.9806201. [91, 92]

[Grossman'09] R. L. Grossman, "The Case for Cloud Computing", in *IT Professional*, vol. 11, no. 2, pp. 23–27, 2009, doi: 10.1109/MITP.2009.40. [2]

[Gupta'18] U. Gupta, M. Babu, R. Ayoub, M. Kishinevsky, F. Paterna, S. Gumussoy, U. Y. Ogras, "An Online Learning Methodology for Performance Modeling of Graphics Processors", in *IEEE Transactions on Computers*, vol. 67, no. 12, pp. 1677–1691, 2018, doi:10.1109/TC.2018.2840710. [93, 95]

[Gupta'20] M. Gupta, L. Bhargava, I. Sreedevi, "Dynamic Voltage Frequency Scaling in Multi-core Systems using Adaptive Regression Model", in *2020 Fourth International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, pp. 1201–1206, 2020, doi:10.1109/I-SMAC49090.2020.9243505. [90, 92]

[Hadsell'20] R. Hadsell, D. Rao, A. A. Rusu, R. Pascanu, "Embracing change: Continual learning in deep neural networks", in *Trends in cognitive sciences*, vol. 24, no. 12, pp. 1028–1040, 2020. [76]

[Hammouda'17] M. B. Hammouda, P. Coussy, L. Lagadec, "A Unified Design Flow to Automatically Generate On-Chip Monitors During High-Level Synthesis of Hardware Accelerators", in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 3, pp. 384–397, 2017, doi: 10.1109/TCAD.2016.2587278. [32, 33]

[Hara'08] Y. Hara, H. Tomiyama, S. Honda, H. Takada, K. Ishii, "CHStone: A benchmark program suite for practical C-based high-level synthesis", in *2008 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1192–1195, 2008, doi: 10.1109/ISCAS.2008.4541637. [55, 56]

[Harris'88] C. Harris, M. Stephens, et al., "A combined corner and edge detector", in *Alvey vision conference*, vol. 15, pp. 10–5244, Manchester, UK, 1988. [188]

[He'20] J. He, R. Mao, Z. Shao, F. Zhu, "Incremental Learning in Online Scenario", in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020. [92]

[Heinz'21] C. Heinz, J. Hofmann, J. Korinth, L. Sommer, L. Weber, A. Koch, "The TaPaSCo Open-Source Toolflow", in *Journal of Signal Processing Systems*, vol. 93, pp. 1–19, 05 2021, doi: 10.1007/s11265-021-01640-8. [26]

[Heinz'24] C. Heinz, T. Kalkhof, Y. Lavan, A. Koch, "TaPaSCo-AIE: An Open-Source Framework for Streaming-Based Heterogeneous Acceleration Using AMD AI Engines", in *2024 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 155–161, 2024, doi: 10.1109/IPDPSW63119.2024.00041. [26]

[Hennessy'17] J. L. Hennessy, D. A. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann, 6th ed., 2017, ISBN 978-0-12-811905-1. [2]

[Hilman'18] M. H. Hilman, M. A. Rodriguez, R. Buyya, "Task Runtime Prediction in Scientific Workflows Using an Online Incremental Learning Approach", in *2018 IEEE/ACM 11th International Conference on Utility and Cloud Computing (UCC)*, pp. 93–102, 2018, doi:10.1109/UCC.2018.00018. [93, 95]

[Hormozi'12] E. Hormozi, H. Hormozi, M. K. Akbari, M. S. Javan, "Using of Machine Learning into Cloud Environment (A Survey): Managing and Scheduling of Resources in Cloud Systems", in *2012 Seventh International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*, pp. 363–368, 2012, doi: 10.1109/3PGCIC.2012.69. [91]

[Houssein'21] E. H. Houssein, A. G. Gad, Y. M. Wazery, P. N. Suganthan, "Task Scheduling in Cloud Computing based on Meta-heuristics: Review, Taxonomy, Open Challenges, and Future Trends", in *Swarm and Evolutionary Computation*, vol. 62, p. 100 841, 2021, ISSN 2210-6502, doi: https://doi.org/10.1016/j.swevo.2021.100841. [17, 18, 19, 138, 142]

[Hulten'01] G. Hulten, L. Spencer, P. Domingos, "Mining time-changing data streams", in *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '01, p. 97–106, Association for Computing Machinery, New York, NY, USA, 2001, ISBN 158113391X, doi: 10.1145/502512.502529. [107]

[Intel'24] Intel, "Quartus® Prime Pro Edition User Guide", UG-20139, 2024. [32]

[Iorio'23] M. Iorio, F. Risso, A. Palesandro, L. Camiciotti, A. Manzalini, "Computing Without Borders: The Way Towards Liquid Computing", in *IEEE Transactions on Cloud Computing*, vol. 11, no. 3, pp. 2820–2838, 2023, doi:10.1109/TCC.2022.3229163. [52]

[Ismail'11] A. Ismail, L. Shannon, "FUSE: Front-End User Framework for O/S Abstraction of Hardware Accelerators", in *2011 IEEE 19th Annual International Symposium on Field-Programmable Custom Computing Machines*, pp. 170–177, 2011, doi: 10.1109/FCCM.2011.48. [27, 29]

[Jacobsen'15] M. Jacobsen, D. Richmond, M. Hogains, R. Kastner, "RIFFA 2.1: A Reusable Integration Framework for FPGA Accelerators", in , vol. 8, no. 4, Sep. 2015, ISSN 1936-7406, doi: 10.1145/2815631. [26, 31]

[Jara-Berrocal'10] A. Jara-Berrocal, A. Gordon-Ross, "VAPRES: A Virtual Architecture for Partially Reconfigurable Embedded Systems", in *2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010)*, pp. 837–842, 2010, doi: 10.1109/DATE.2010.5456934. [27, 29]

[Javed'18] A. Javed, K. Heljanko, A. Buda, K. Främling, "CEFIoT: A fault-tolerant IoT architecture for edge and cloud", in *2018 IEEE 4th World Forum on Internet of Things (WF-IoT)*, pp. 813–818, 2018, doi:10.1109/WF-IoT.2018.8355149. [3]

[Jawarneh'19] I. M. A. Jawarneh, P. Bellavista, F. Bosi, L. Foschini, G. Martuscelli, R. Montanari, A. Palopoli, "Container Orchestration Engines: A Thorough Functional and Performance Comparison", in *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, pp. 1–6, 2019, doi:10.1109/ICC.2019.8762053. [16]

[Kamisetty'23] A. Kamisetty, M. Rodriguez, S. Kothapalli, "Microservices vs. Monoliths: Comparative Analysis for Scalable Software Architecture Design", in *Engineering International*, vol. 11, pp. 99–112, 12 2023, doi:10.18034/ei.v11i2.734. [29]

[Kandula'09] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, R. Chaiken, "The nature of data center traffic: measurements & analysis", in *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement*, IMC '09, p. 202–208, Association for Computing Machinery, New York, NY, USA, 2009, ISBN 9781605587714, doi:10.1145/1644893.1644918. [78]

[Karabulut'25] E. Karabulut, A. A. Malik, A. Awad, A. Aysu, "THEMIS: Time, Heterogeneity, and Energy Minded Scheduling for Fair Multi-Tenant Use in FPGAs", in *IEEE Transactions on Computers*, pp. 1–14, 2025, doi:10.1109/TC.2025.3566874. [140, 141, 142]

[Kennedy'95] J. Kennedy, R. Eberhart, "Particle swarm optimization", in *Proceedings of ICNN'95 - International Conference on Neural Networks*, vol. 4, pp. 1942–1948 vol.4, 1995, doi:10.1109/ICNN.1995.488968. [145]

[Khalyeyev'23] D. Khalyeyev, T. Bureš, P. Hnětynka, "Towards Characterization of Edge-Cloud Continuum", in *Software Architecture. ECSA 2022 Tracks and Workshops*, pp. 215–230, Springer International Publishing, Cham, 2023, ISBN 978-3-031-36889-9, doi:10.1007/978-3-031-36889-9_16. [3, 15]

[Khazraee'17] M. Khazraee, L. Zhang, L. Vega, M. B. Taylor, "Moonwalk: NRE Optimization in ASIC Clouds", in *SIGPLAN Not.*, vol. 52, no. 4, p. 511–526, Apr. 2017, ISSN 0362-1340, doi:10.1145/3093336.3037749. [8]

[Khetawat'24] H. Khetawat, F. Mueller, "Workload Scheduling on Heterogeneous Devices", in *ISC High Performance 2024 Research Paper*

Proceedings (39th International Conference), pp. 1–11, 2024, doi:10.23919/ISC.2024.10528933. [140, 142]

[Kingman'92] J. F. C. Kingman, *Poisson processes*, vol. 3, Clarendon Press, 1992. [78]

[Koch'12] D. Koch, *Partial reconfiguration on FPGAs: architectures, tools and applications*, vol. 153, Springer Science & Business Media, 2012, ISBN 978-1-4614-1225-0, doi:10.1007/978-1-4614-1225-0. [2, 13]

[Krommydas'16] K. Krommydas, W.-c. Feng, C. D. Antonopoulos, N. Bellas, "OpenDwarfs: Characterization of Dwarf-Based Benchmarks on Fixed and Reconfigurable Architectures", in , vol. 85, no. 3, pp. 373–392, 2016, ISSN 1939-8115, doi:10.1007/s11265-015-1051-z. [55]

[Lallet'18] J. Lallet, A. Enrici, A. Saffar, "FPGA-Based System for the Acceleration of Cloud Microservices", in *2018 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)*, pp. 1–5, 2018, doi:10.1109/BMSB.2018.8436912. [30, 31]

[Larsson'20] L. Larsson, H. Gustafsson, C. Klein, E. Elmroth, "Decentralized Kubernetes Federation Control Plane", in *2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC)*, pp. 354–359, 2020, doi:10.1109/UCC48980.2020.00056. [52]

[Lee'97] C. Lee, M. Potkonjak, W. Mangione-Smith, "MediaBench: a tool for evaluating and synthesizing multimedia and communications systems", in *Proceedings of 30th Annual International Symposium on Microarchitecture*, pp. 330–335, 1997, doi:10.1109/MICRO.1997.645830. [54]

[Lee'15] J. C. Lee, R. Lysecky, "System-Level Observation Framework for Non-Intrusive Runtime Monitoring of Embedded Systems", in , vol. 20, no. 3, Jun. 2015, ISSN 1084-4309, doi:10.1145/2717310, https://doi.org/10.1145/2717310. [32, 33]

[Leeser'21] M. Leeser, S. Handagala, M. Zink, "FPGAs in the Cloud", in *Computing in Science & Engineering*, vol. 23, no. 6, pp. 72–76, 2021, doi:10.1109/MCSE.2021.3127288. [3, 29]

[Leutenegger'11] S. Leutenegger, M. Chli, R. Y. Siegwart, "BRISK: Binary Robust invariant scalable keypoints", in *2011 International Conference on Computer Vision*, pp. 2548–2555, 2011, doi:10.1109/ICCV.2011.6126542. [189]

[Li'02] Z. Li, S. Hauck, "Configuration prefetching techniques for partial reconfigurable coprocessor with relocation and defragmentation", in *Proceedings of the 2002 ACM/SIGDA Tenth International Symposium on Field-Programmable Gate Arrays*, FPGA '02, p. 187–195, Association for Computing Machinery, New York, NY, USA, 2002, ISBN 1581134525, doi: 10.1145/503048.503076. [140]

[Li'17] C. V. Li, V. Petrucci, D. Mossé, "Exploring Machine Learning for Thread Characterization on Heterogeneous Multiprocessors", in , vol. 51, no. 1, p. 113–123, Sep. 2017, ISSN 0163-5980, doi: 10.1145/3139645.3139664. [90, 92]

[Li'22] B. Li, X. Zhang, H. You, Z. Qi, Y. Zhang, "Machine Learning Based Framework for Fast Resource Estimation of RTL Designs Targeting FPGAs", in *ACM Trans. Des. Autom. Electron. Syst.*, vol. 28, no. 2, Dec. 2022, ISSN 1084-4309, doi: 10.1145/3555047. [91, 92]

[Lombardo'12] M. Lombardo, J. Camarero, J. Valverde, J. Portilla, E. de la Torre, T. Riesgo, "Power management techniques in an FPGA-based WSN node for high performance applications", in *7th International Workshop on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC)*, pp. 1–8, 2012, doi:10.1109/ReCoSoC.2012.6322888. [13]

[Long'20] X. Long, B. Liu, F. Jiang, Q. Zhang, X. Zhi, "FPGA virtualization deployment based on Docker container technology", in *2020 5th International Conference on Mechanical, Control and Computer Engineering (ICMCCE)*, pp. 473–476, 2020, doi: 10.1109/ICMCCE51767.2020.00109. [30, 31]

[Lopes'20] A. S. B. Lopes, M. M. Pereira, "A Machine Learning Approach to Accelerating DSE of Reconfigurable Accelerator Systems", in *2020 33rd Symposium on Integrated Circuits and Systems Design (SBCCI)*, pp. 1–6, 2020, doi: 10.1109/SBCCI50935.2020.9189899. [91, 92]

[Lowe'99] D. Lowe, "Object recognition from local scale-invariant features", in *Proceedings of the Seventh IEEE International Conference on Computer Vision*, vol. 2, pp. 1150–1157 vol.2, 1999, doi:10.1109/ICCV.1999.790410. [189]

[M. Makrani'19] H. M. Makrani, F. Farahmand, H. Sayadi, S. Bondi, S. M. Pudukotai Dinakarrao, H. Homayoun, S. Rafatirad, "Pyramid: Machine Learning Framework to Estimate the Optimal Timing and Resource Usage of a High-Level Synthesis

Design", in *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*, pp. 397–403, 2019, doi:10.1109/FPL.2019.00069. [91, 92]

[Majumder'21] P. Majumder, J. Huang, S. Kim, A. Muzahid, D. Siegers, C.-C. Tsai, E. J. Kim, "Continual Learning Approach for Improving the Data and Computation Mapping in Near-Memory Processing System", in *arXiv preprint arXiv:2104.13671*, 2021. [93, 95]

[Manca'25] F. Manca, F. Ratto, F. Palumbo, "ONNX-To-Hardware Design Flow for Adaptive Neural-Network Inference on FPGAs", in L. Carro, F. Regazzoni, C. Pilato, editors, *Embedded Computer Systems: Architectures, Modeling, and Simulation*, pp. 85–96, Springer Nature Switzerland, Cham, 2025, ISBN 978-3-031-78380-7. [58]

[Mandal'20] S. K. Mandal, G. Bhat, J. R. Doppa, P. P. Pande, U. Y. Ogras, "An Energy-aware Online Learning Framework for Resource Management in Heterogeneous Platforms", in *ACM Trans. Des. Autom. Electron. Syst.*, vol. 25, no. 3, May 2020, ISSN 1084-4309, doi:10.1145/3386359. [94, 95]

[Matthews'10] E. Matthews, L. Shannon, A. Fedorova, "A configurable framework for investigating workload execution", in *2010 International Conference on Field-Programmable Technology*, pp. 409–412, 2010, doi:10.1109/FPT.2010.5681447. [32]

[Mehrabi'22] A. Mehrabi, D. J. Sorin, B. C. Lee, "Spatiotemporal Strategies for Long-Term FPGA Resource Management", in *2022 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 198–209, 2022, doi:10.1109/ISPASS55109.2022.00026. [140, 141, 142]

[Mei'04] B. Mei, S. Vernalde, D. Verkest, R. Lauwereins, "Design methodology for a tightly coupled VLIW/reconfigurable matrix architecture: a case study", in *Proceedings Design, Automation and Test in Europe Conference and Exhibition*, vol. 2, pp. 1224–1229 Vol.2, 2004, doi:10.1109/DATE.2004.1269063. [9]

[Mirka'20] M. Mirka, G. Sassatelli, A. Gamatié, "Online Learning for Dynamic Control of OpenMP Workloads", in *2020 9th International Conference on Modern Circuits and Systems Technologies (MOCAST)*, pp. 1–6, 2020, doi:10.1109/MOCAST49295.2020.9200292. [93, 95]

[Mittal'17] S. Mittal, N. Negi, R. Chauhan, "Integration of edge computing with cloud computing", in *2017 International Conference on Emerging Trends in Computing and Communication Technologies (ICETCCT)*, pp. 1–6, 2017, doi: 10.1109/ICETCCT.2017.8280340. [3]

[Montiel'21] J. Montiel, M. Halford, S. M. Mastelini, G. Bolmier, R. Sourty, R. Vaysse, A. Zouitine, H. M. Gomes, J. Read, T. Abdessalem, A. Bifet, "River: machine learning for streaming data in Python", in *Journal of Machine Learning Research*, vol. 22, no. 110, pp. 1–8, 2021, http://jmlr.org/papers/v22/20-1380.html. [100, 107]

[Moore'64] G. Moore, "The future of integrated electronics", in *Understanding Moore's Law: Four Decades of Innovation*, pp. 37–55, 1964. [1]

[Moore'06] G. E. Moore, "Cramming more components onto integrated circuits, Reprinted from Electronics, volume 38, number 8, April 19, 1965, pp.114 ff.", in *IEEE Solid-State Circuits Society Newsletter*, vol. 11, no. 3, pp. 33–35, 2006, doi:10.1109/N-SSC.2006.4785860. [1]

[Muja'09] M. Muja, D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration.", in *VISAPP (1)*, vol. 2, no. 331-340, p. 2, 2009. [189]

[Nadimpalli'16] P. K. Nadimpalli, S. K. Roy, "An efficient FPGA-based function profiler for embedded system applications", in *2016 20th International Symposium on VLSI Design and Test (VDAT)*, pp. 1–6, 2016, doi:10.1109/ISVDAT.2016.8064857. [32]

[Najem'17] M. Najem, P. Benoit, M. El Ahmad, G. Sassatelli, L. Torres, "A Design-Time Method for Building Cost-Effective Run-Time Power Monitoring", in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 7, pp. 1153–1166, 2017, doi:10.1109/TCAD.2016.2614347. [32, 33]

[Nurvitadhi'16] E. Nurvitadhi, J. Sim, D. Sheffield, A. Mishra, S. Krishnan, D. Marr, "Accelerating recurrent neural networks in analytics servers: Comparison of FPGA, CPU, GPU, and ASIC", in *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, pp. 1–4, 2016, doi: 10.1109/FPL.2016.7577314. [2]

[Ojika'19] D. Ojika, A. Gordon-Ross, H. Lam, B. Patel, "FaaM: FPGA-as-a-Microservice - A Case Study for Data Compression", in *EPJ Web Conf.*, vol. 214, p. 07 029, 2019, doi: 10.1051/epjconf/201921407029. [30, 31]

[O'Neal'18] K. O'Neal, P. Brisk, "Predictive Modeling for CPU, GPU, and FPGA Performance and Power Consumption: A Survey", in *2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pp. 763–768, 2018, doi:10.1109/ISVLSI.2018.00143. [90, 91]

[Pagani'20] S. Pagani, P. D. S. Manoj, A. Jantsch, J. Henkel, "Machine Learning for Power, Energy, and Thermal Management on Multicore Processors: A Survey", in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 1, pp. 101–116, 2020, doi: 10.1109/TCAD.2018.2878168. [90]

[Palumbo'24] F. Palumbo, et al., "MYRTUS: Multi-layer 360° dYnamic orchestration and interopeRable design environmenT for compute-continUum Systems", in *Proceedings of the 21st ACM International Conference on Computing Frontiers: Workshops and Special Sessions*, CF '24 Companion, p. 101–106, Association for Computing Machinery, New York, NY, USA, 2024, ISBN 9798400704925, doi:10.1145/3637543.3654618. [xxiii, 6]

[Papadimitriou'98] C. H. Papadimitriou, K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Dover Publications, July 1998, ISBN 0486402584. [17]

[Parkhurst'06] J. Parkhurst, J. Darringer, B. Grundmann, "From single core to multi-core: preparing for a new exponential", in *Proceedings of the 2006 IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '06, p. 67–72, Association for Computing Machinery, New York, NY, USA, 2006, ISBN 1595933891, doi: 10.1145/1233501.1233516. [2]

[Patrigeon'18] G. Patrigeon, P. Benoit, L. Torres, "FPGA-Based Platform for Fast Accurate Evaluation of Ultra Low Power SoC", in *2018 28th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, pp. 123–128, 2018, doi:10.1109/PATMOS.2018.8464173. [32]

[Paul'24] R. Paul, M. Danelutto, "Power Aware Scheduling of Tasks on FPGAs in Data Centers", in *2024 32nd Euromicro International Conference on Parallel, Distributed and*

Network-Based Processing (PDP), pp. 148–152, 2024, doi: 10.1109/PDP62718.2024.00028. [140, 141, 142]

[Pham'16] N. K. Pham, A. Kumar, A. K. Singh, M. M. A. Khin, "Leakage aware resource management approach with machine learning optimization framework for partially reconfigurable architectures", in *Microprocessors and Microsystems*, vol. 47, pp. 231–243, 2016, ISSN 0141-9331, doi: https://doi.org/10.1016/j.micpro.2016.09.012. [91, 92]

[Pham'19] K. D. Pham, K. Paraskevas, A. Vaishnav, A. Attwood, M. Vesper, D. Koch, "ZUCL 2.0: Virtualised Memory and Communication for ZYNQ UltraScale+ FPGAs", in *FSP Workshop 2019; Sixth International Workshop on FPGAs for Software Programmers*, pp. 1–9, 2019. [26]

[Qasaimeh'19] M. Qasaimeh, K. Denolf, J. Lo, K. Vissers, J. Zambreno, P. H. Jones, "Comparing Energy Efficiency of CPU, GPU and FPGA Implementations for Vision Kernels", in *2019 IEEE International Conference on Embedded Software and Systems (ICESS)*, pp. 1–8, 2019, doi:10.1109/ICESS.2019.8782524. [2]

[Ramezani'20] R. Ramezani, "A prefetch-aware scheduling for FPGA-based multi-task graph systems", in *The Journal of Supercomputing*, vol. 76, no. 9, pp. 7140–7160, 2020. [140, 142]

[Rapp'21] M. Rapp, A. Pathania, T. Mitra, J. Henkel, "Neural Network-Based Performance Prediction for Task Migration on S-NUCA Many-Cores", in *IEEE Transactions on Computers*, vol. 70, no. 10, pp. 1691–1704, 2021, doi:10.1109/TC.2020.3023022. [90, 92]

[Reagen'14] B. Reagen, R. Adolf, Y. S. Shao, G.-Y. Wei, D. Brooks, "MachSuite: Benchmarks for accelerator design and customized architectures", in *2014 IEEE International Symposium on Workload Characterization (IISWC)*, pp. 110–119, 2014, doi: 10.1109/IISWC.2014.6983050. [xxvii, 55, 56, 58]

[Rodríguez'20] A. Rodríguez, *A Framework to Support Run-Time Adaptation in Reconfigurable Multi-Accelerator Systems*, Ph.D. thesis, E.T.S.I. Industriales (UPM), 2020, doi: 10.20868/UPM.thesis.65388. [xxiii, 14]

[Rodríguez'18] A. Rodríguez, J. Valverde, J. Portilla, A. Otero, T. Riesgo, E. De la Torre, "FPGA-Based High-Performance Embedded Systems for Adaptive Edge Computing in Cyber-Physical Systems: The

ARTICo3 Framework", in *Sensors*, vol. 18, no. 6, 2018, ISSN 1424-8220, doi:10.3390/s18061877. [13, 28, 29, 44, 58]

[Roorda'23] E. Roorda, S. J. Wilton, "Online Training from Streaming Data with Concept Drift on FPGAs", in *2023 24th International Symposium on Quality Electronic Design (ISQED)*, pp. 1–8, 2023, doi:10.1109/ISQED57927.2023.10129312. [94, 95]

[Rublee'11] E. Rublee, V. Rabaud, K. Konolige, G. Bradski, "ORB: An efficient alternative to SIFT or SURF", in *2011 International Conference on Computer Vision*, pp. 2564–2571, 2011, doi: 10.1109/ICCV.2011.6126544. [189]

[Sadiku'14] M. N. Sadiku, S. M. Musa, O. D. Momoh, "Cloud Computing: Opportunities and Challenges", in *IEEE Potentials*, vol. 33, no. 1, pp. 34–36, 2014, doi:10.1109/MPOT.2013.2279684. [2]

[Sarlin'20] P.-E. Sarlin, D. DeTone, T. Malisiewicz, A. Rabinovich, "SuperGlue: Learning Feature Matching With Graph Neural Networks", in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020. [189]

[Satyanarayanan'17] M. Satyanarayanan, "The Emergence of Edge Computing", in *Computer*, vol. 50, no. 1, pp. 30–39, 2017, doi: 10.1109/MC.2017.9. [2]

[Sau'21] C. Sau, T. Fanni, C. Rubattu, L. Raffo, F. Palumbo, "The Multi-Dataflow Composer tool: An open-source tool suite for optimized coarse-grain reconfigurable hardware accelerators and platform design", in *Microprocessors and Microsystems*, vol. 80, p. 103 326, 2021, ISSN 0141-9331, doi: https://doi.org/10.1016/j.micpro.2020.103326, https://www.sciencedirect.com/science/article/pii/S0141933120304853. [44]

[Shi'94] J. Shi, Tomasi, "Good features to track", in *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pp. 593–600, 1994, doi:10.1109/CVPR.1994.323794. [188]

[Shi'16] W. Shi, J. Cao, Q. Zhang, Y. Li, L. Xu, "Edge Computing: Vision and Challenges", in *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016, doi:10.1109/JIOT.2016.2579198. [76]

[Singh'92] J. P. Singh, W.-D. Weber, A. Gupta, "SPLASH: Stanford parallel applications for shared-memory", in *ACM SIGARCH Computer Architecture News*, vol. 20, no. 1, pp. 5–44, 1992. [54]

[Singh'00] H. Singh, M.-H. Lee, G. Lu, F. Kurdahi, N. Bagherzadeh, E. Chaves Filho, "MorphoSys: an integrated reconfigurable system for data-parallel and computation-intensive applications", in *IEEE Transactions on Computers*, vol. 49, no. 5, pp. 465–481, 2000, doi:10.1109/12.859540. [9]

[Singh'21] H. Singh, S. Tyagi, P. Kumar, S. S. Gill, R. Buyya, "Metaheuristics for scheduling of heterogeneous tasks in cloud computing environments: Analysis, performance evaluation, and future directions", in *Simulation Modelling Practice and Theory*, vol. 111, p. 102 353, 2021, ISSN 1569-190X, doi: https://doi.org/10.1016/j.simpat.2021.102353. [145]

[Smith'24] M. Smith, L. Zhao, J. Cordova, X. Jiang, M. Ebrahimi, "Machine Learning-Based Energy-efficient Workload Management for Data Centers", in *2024 IEEE 21st Consumer Communications and Networking Conference (CCNC)*, pp. 799–802, 2024, doi: 10.1109/CCNC51664.2024.10454842. [141, 142]

[Sousa'22] L. M. Sousa, N. Paulino, J. C. Ferreira, J. Bispo, "A Flexible HLS Hoeffding Tree Implementation for Runtime Learning on FPGA", in *2022 IEEE 21st Mediterranean Electrotechnical Conference (MELECON)*, pp. 972–977, 2022, doi: 10.1109/MELECON53508.2022.9843092. [94, 95]

[SPEC] "Standard Performance Evaluation Corporation Benchmarks", https://www.spec.org/. Accessed: April 10, 2025. [54]

[Tian'21] Z. Tian, L. Chen, X. Li, J. Feng, J. Xu, "Multi-Core Power Management through Deep Reinforcement Learning", in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, 2021, doi:10.1109/ISCAS51556.2021.9401447. [90, 92]

[Tian'23] X. Tian, X. Li, J. Zhang, Z. Zhao, C. Wang, X. Wang, J. Wang, "An Online Incremental Learning Framework for HPC Job Power Consumption Prediction", in *Proceedings of the 2023 7th International Conference on High Performance Compilation, Computing and Communications*, HP3C '23, p. 176–183, Association for Computing Machinery, New York, NY, USA, 2023, ISBN 9781450399883, doi:10.1145/3606043.3606068. [93, 95]

[Tianyang'21] L. Tianyang, Z. Fan, G. Wei, S. Mingqian, C. Li, "A Survey: FPGA-Based Dynamic Scheduling of Hardware Tasks", in

Chinese Journal of Electronics, vol. 30, no. 6, pp. 991–1007, 2021, doi:https://doi.org/10.1049/cje.2021.07.021. [138, 140]

[Trajković'98] M. Trajković, M. Hedley, "Fast corner detection", in *Image and Vision Computing*, vol. 16, no. 2, pp. 75–87, 1998, ISSN 0262-8856, doi:https://doi.org/10.1016/S0262-8856(97)00056-5. [189]

[Vaishnav'18] A. Vaishnav, K. D. Pham, D. Koch, "A Survey on FPGA Virtualization", in *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*, pp. 131–1317, 2018, doi:10.1109/FPL.2018.00031. [32]

[Vaishnav'19] A. Vaishnav, K. D. Pham, D. Koch, "Heterogeneous Resource-Elastic Scheduling for CPU+FPGA Architectures", in *Proceedings of the 10th International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies*, HEART '19, Association for Computing Machinery, New York, NY, USA, 2019, ISBN 9781450372558, doi:10.1145/3337801.3337819. [140, 142]

[Vaishnav'20] A. Vaishnav, K. D. Pham, J. Powell, D. Koch, "FOS: A Modular FPGA Operating System for Dynamic Workloads", in *ACM Trans. Reconfigurable Technol. Syst.*, vol. 13, no. 4, Sep. 2020, ISSN 1936-7406, doi:10.1145/3405794. [140, 141, 142]

[Valente'21] G. Valente, T. Fanni, C. Sau, T. D. Mascio, L. Pomante, F. Palumbo, "A Composable Monitoring System for Heterogeneous Embedded Platforms", in *ACM Trans. Embed. Comput. Syst.*, vol. 20, no. 5, Jul. 2021, ISSN 1539-9087, doi:10.1145/3461647, https://doi.org/10.1145/3461647. [32, 33]

[Vipin'14] K. Vipin, S. A. Fahmy, "DyRACT: A partial reconfiguration enabled accelerator and test platform", in *2014 24th International Conference on Field Programmable Logic and Applications (FPL)*, pp. 1–7, 2014, doi:10.1109/FPL.2014.6927507. [26]

[Xilinx'19a] Xilinx, "AXI HBICAP Product Guide", PG-349, 2019. [35]

[Xilinx'19b] Xilinx, "Zynq UltraScale+ test Software Developer Guide", UG-1137, 2019. [71]

[Xilinx'21] Xilinx, "System Integrated Logic Analyzer Product Guide", PG-261, 2021. [32]

[Xilinx'23a] Xilinx, "Alveo Card Management Solution Subsystem Product Guide", PG348, 2023. [62]

[Xilinx'23b] Xilinx, "Kria SOM App Store Applications Developer Deployment Guide for Ubuntu", UG-1630, 2023. [32]

[Xilinx'23c] Xilinx, "Vitis Unified Software Platform Documentation: Application Acceleration Development", UG-1393, 2023. [44]

[Xilinx'23d] Xilinx, "ZCU102 Evaluation Board User Guide", UG-1182, 2023. [41]

[Xilinx'23e] Xilinx, "Zynq 7000 SoC Technical Reference Manual", UG-585, 2023. [35]

[Xilinx'24] Xilinx, "DMA/Bridge Subsystem for PCI Express Product Guide", PG-195, 2024. [34]

[Xilinx'25a] Xilinx, "AXI HWICAP Product Guide", PG-134, 2025. [35]

[Xilinx'25b] Xilinx, "Kria KV260 Vision AI Starter Kit User Guide", UG-1089, 2025. [41]

[Xu'22] C. Xu, S. Jiang, G. Luo, G. Sun, N. An, G. Huang, X. Liu, "The Case for FPGA-Based Edge Computing", in *IEEE Transactions on Mobile Computing*, vol. 21, no. 7, pp. 2610–2619, 2022, doi: 10.1109/TMC.2020.3041781. [3, 29]

[Yang'10] X.-S. Yang, *Engineering Optimization: An Introduction with Metaheuristic Applications*, Wiley Publishing, 1st ed., 2010, ISBN 0470582464, doi:10.5555/1892063. [17, 138]

[Zacarias'21] F. V. Zacarias, V. Petrucci, R. Nishtala, P. Carpenter, D. Mossé, "Intelligent colocation of HPC workloads", in *Journal of Parallel and Distributed Computing*, vol. 151, pp. 125–137, 2021, ISSN 0743-7315, doi: https://doi.org/10.1016/j.jpdc.2021.02.010. [141, 142]

[Zaruba'19] F. Zaruba, L. Benini, "The cost of application-class processing: Energy and performance analysis of a Linux-ready 1.7-GHz 64-bit RISC-V core in 22-nm FDSOI technology", in *IEEE Transactions on VLSI Systems*, vol. 27, no. 11, 2019. [65]

[Zhou'18] Y. Zhou, U. Gupta, S. Dai, R. Zhao, N. Srivastava, H. Jin, J. Featherston, Y.-H. Lai, G. Liu, G. A. Velasquez, W. Wang, Z. Zhang, "Rosetta: A Realistic High-Level Synthesis Benchmark Suite for Software Programmable FPGAs", in *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '18, p. 269–278, Association for Computing Machinery, New York, NY, USA, 2018, ISBN 9781450356145, doi:10.1145/3174243.3174255. [55, 56]

[Zoni'18] D. Zoni, L. Cremona, A. Cilardo, M. Gagliardi, W. Fornaciari, "PowerTap: All-digital power meter modeling for run-time power monitoring", in *Microprocessors and Microsystems*, vol. 63, pp. 128–139, 2018, ISSN 0141-9331, doi: https://doi.org/10.1016/j.micpro.2018.07.007, https://www.sciencedirect.com/science/article/pii/S0141933118302308. 32, 33