UNIVERSITÀ
DEGLI STUDI
DI TRIESTE

UNIVERSITÀ DEGLI STUDI DI TRIESTE

Joint-supervision with Universidad Nacional de San Luis

XXXV Ciclo del Dottorato di Ricerca in Ingegneria Industriale e dell'Informazione

Doctorado en Ciencias de la Computación

PO FRIULI VENEZIA GIULIA - FONDO SOCIALE EUROPEO 2014/2020

SoC-based FPGA architecture for image analysis and other highly demanding applications

---

Settore scientifico-disciplinare: ING-INF/01 Electronics

Dottoranda:	Romina Soledad Molina	
Coordinatore:	Alberto Tessarolo	
Supervisori di tesi:	Giovanni Ramponi	
	Verónica Gil Costa	
Co-supervisore di tesi:	María Liz Crespo	

Anno accademico 2021/2022

"You will begin to touch heaven, Jonathan,
at the moment that you touch perfect speed.
And that isn't flying a thousand miles an hour,
or a million, or flying at the speed of light.
Because any number is a limit,
and perfection doesn't have limits.
Perfect speed, my son, is being there."

— Jonathan Livingston Seagull - Richard Bach

Acknowledgment

Grazie a Dio perché la sua energia è sempre presente, in ogni passo.

Grazie ai miei supervisori, Gianni, Liz, e Verónica, che sono stati presenti durante tutto questo tempo, guidando la barca che oggi mi ha permesso di raggiungere la meta di questo viaggio. Grazie per aver nutrito il mio cammino con la sua saggezza e la lora buona qualità umana. Grazie anche ad Andrés, che mi ha dato il suo sostegno in questa parte del viaggio.

Grazie alle istituzioni che mi hanno permesso di sviluppare il mio dottorato: Università degli Studi di Trieste, The Abdus Salam International Centre for Theoretical Physics, Consiglio Nazionale delle Ricerche, Universidad Nacional de San Luis.

Grazie eterne alla mia famiglia, perché anche a distanza fisica sono sempre presenti, con tutto l'amore, il supporto, e la comprensione. Grazie per essere sempre al mio fianco. Vi volgio tanto bene!

Andrea una volta mi ha detto che i supervisori sono come il capitano della nave, ci guidano in questo processo di apprendimento che attraversa ogni ambito della nostra vita. Grazie infinite essere presente, per avermi permesso d'imparare dalla tua saggezza.

Grazie Raffaele, Franco e Salvatore, per avermi accompagnato in questi anni.

Grazie a Iván, Werner, Charm Loong, Luis, Bruno, Maynor, e Agustín per avermi accompagnato in tutti questi anni, per l'amicizia, le risate, i momenti di lavoro, e quelli di relax.

Grazie a tutti i miei amici dell'anima che vibrano nell'eternità dell'esistenza.

Infinitamente, grazie al mare!

Romina Soledad Molina

Summary

Nowadays, the development of algorithms focuses on performance-efficient and energy-efficient computations. Technologies such as field programmable gate array (FPGA) and system on chip (SoC) based on FPGA (FPGA/SoC) have shown their ability to accelerate intensive computing applications while saving power consumption, owing to their capability of high parallelism and reconfiguration of the architecture.

Currently, the existing design cycles for FPGA/SoC are time-consuming, owing to the complexity of the architecture. Therefore, to address the gap between applications and FPGA/SoC architectures and to obtain an efficient hardware design for image analysis and highly demanding applications using the high-level synthesis tool, two complementary strategies are considered: ad-hoc techniques and performance estimator.

Regarding ad-hoc techniques, three highly demanding applications were accelerated through HLS tools: pulse shape discriminator for cosmic rays, automatic pest classification, and re-ranking for information retrieval, emphasizing the benefits when this type of applications are traversed by compression techniques when targeting FPGA/SoC devices.

Furthermore, a comprehensive performance estimator for hardware acceleration is proposed in this thesis to effectively predict the resource utilization and latency for FPGA/SoC, building a bridge between the application and architectural domains. The tool integrates analytical models for performance prediction, and a design space explorer (DSE) engine for providing high-level insights to hardware developers, composed of two independent sub-engines: DSE based on single-objective optimization and DSE based on evolutionary multi-objective optimization.

Riassunto

Al giorno d'oggi, lo sviluppo di algoritmi si concentra su calcoli efficienti in termini di prestazioni ed efficienza energetica. Tecnologie come il field programmable gate array (FPGA) e il system on chip (SoC) basato su FPGA (FPGA/SoC) hanno dimostrato la loro capacità di accelerare applicazioni di calcolo intensive risparmiando al contempo il consumo energetico, grazie alla loro capacità di elevato parallelismo e riconfigurazione dell'architettura.

Attualmente, i cicli di progettazione esistenti per FPGA/SoC sono lunghi, a causa della complessità dell'architettura. Pertanto, per colmare il divario tra le applicazioni e le architetture FPGA/SoC e ottenere un design hardware efficiente per l'analisi delle immagini e altri applicazioni altamente demandanti utilizzando lo strumento di sintesi di alto livello, vengono prese in considerazione due strategie complementari: tecniche ad hoc e stima delle prestazioni.

Per quanto riguarda le tecniche ad-hoc, tre applicazioni molto impegnative sono state accelerate attraverso gli strumenti HLS: discriminatore di forme di impulso per i raggi cosmici, classificazione automatica degli insetti e re-ranking per il recupero delle informazioni, sottolineando i vantaggi quando questo tipo di applicazioni viene attraversato da tecniche di compressione durante il targeting dispositivi FPGA/SoC.

Inoltre, in questa tesi viene proposto uno stimatore delle prestazioni per l'accelerazione hardware per prevedere efficacemente l'utilizzo delle risorse e la latenza per FPGA/SoC, costruendo un ponte tra l'applicazione e i domini architetturali. Lo strumento integra modelli analitici per la previsione delle prestazioni e un motore design space explorer (DSE) per fornire approfondimenti di alto livello agli sviluppatori di hardware, composto da due motori indipendenti: DSE basato sull'ottimizzazione a singolo obiettivo e DSE basato sull'ottimizzazione evolutiva multiobiettivo.

Contents

List of Figures

List of Tables

List of acronyms

A	Area
ADRS	Average distance from reference set
AF	Array factor
ANN	Artificial neural network
AP	Attainable performance
ASIC	Application specific integrated circuit
AUC	Area under the curve
BBO	Black-box optimization
BRAM	Block RAM
BO	Bayesian optimization
BSP	Bulk synchronous parallel
CCM	Collective computing model
CDFG	Control data flow graph
CFD	Computational fluid dynamics
CI	Computational intensity
CLBs	Configurable logic blocks

CNN Convolutional neural network

CR Cosmic rays

CRCW Concurrent read concurrent write

CREW Concurrent read exclusive write

CUDA Compute Unified Device Architecture

D Design space

DAQ Data acquisition systems

DDDG Dynamic data dependence graph

DMA Direct memory access

DNN Deep neural network

DSE Design space exploration

DSP Digital signal processor

EA Evolutionary algorithm

EI Expected improvement

EMO Evolutionary multi-objective optimization

ERCW Exclusive read concurrent write

EREW Exclusive read exclusive write

ERT Empirical Roofline toolkit

FF Flip-Flop

FIR Finite impulse response

FPGA Field programmable gate array

GNN Graph neural networks

GP Gaussian process

HDL Hardware description language

HLS High-level synthesis

HPC High-performance computing

HPM Hierarchical model for parallel computations

HVE Hypervolume error

II Initiation interval

I/O Input/Output

IL Iteration latency

IoT Internet of things

IP Intellectual property

IR Intermediate representation

KD Knowledge distillation

L Latency

L1 Level-1 cache memory

L2 Level-2 cache memory

LAGO The Latin American Giant Observatory

LLVM IR Low-level virtual machine intermediate representation

LUT LookUp Table

ML Machine learning

MLP Multi-layer perceptro

MOOA Multi-objective optimization algorithms

MPSoC Multiprocessor system on chip

NN Neural network

NSGA-II Elitist non-dominated sorting genetic algorithm

P & R Place and route

PC Peak computation

PE Processing element

PF Pareto-optimal frontier

PI Probability of improvement

PMB Peak memory bandwidth

PRAM Parallel random access machine

PSD Pulse shape discriminator

QAP Quantization-aware pruning

QAT Quantization-aware training

QoR Quality of results

QS QuickScorer

RAM Random access machine

ROC Receiver operating characteristics

RTL Register transfer level

SC scalability

SIMD Single Instruction/Multiple Data

SoC System on chip

SPMD Single program multiple data

T	Throughput
TC	Trip count
UF	Unrolling factor
UMH	Uniform Memory Hierarchy Model of Computation
WCD	Water Cherenkov detectors

Chapter 1

Introduction

1.1 Motivation

Nowadays, the development of algorithms focuses on performance-efficient and energy-efficient computations. Technologies such as field programmable gate array (FPGA) and system on chip (SoC) based on FPGA (FPGA/SoC) [2–5] have shown their ability to accelerate intensive computing applications while saving power consumption, owing to their capability of high parallelism and reconfiguration of the architecture.

Several high-level synthesis (HLS) tools [6] have been proposed by vendors and academics, such as Vivado HLS [7], formerly AutoPilot [8], Intel HLS [9], LegUp [10], Bambu [11], and others [12]. These tools facilitate the adoption of FPGAs in different fields, as they allow the creation of a register transfer level (RTL) code from a high level of abstraction. Nevertheless, the efficient use of these technologies usually requires the knowledge of the underlying hardware and code restructuring techniques in the original algorithm [13]. This is a time-consuming task for algorithm designers who want to take advantage of the inherent characteristics of these reconfigurable technologies.

HLS tools support C/C++, SystemC, and OpenCL [14] codes to generate the final RTL code.

This chapter is based on the work published in [1]: **Molina, R.S.**; Gil-Costa, V.; Crespo, M. L; Ramponi, G. (2022) "High-Level Synthesis Hardware Design for FPGA-based Accelerators: Models, Methodologies, and Frameworks". In IEEE Access, vol. 10, pp. 90429-90455, 2022. IEEE.

These tools provide the designer with a detailed report for each algorithmic solution, including information about the estimation of latency, resource utilization (also known as the area occupied), and throughput. The use of directives allows code optimization through parallel techniques, such as loop pipelining, loop unrolling, array partitioning, and array reshaping. For each solution, the designer can specify different combinations of directives; comparing the reports provided by these tools, the best option can be determined according to different performance metrics.

Furthermore, these tools allow a design space exploration (DSE), which involves the evaluation of multiple implementations with different combinations of user design constraints, FPGA features, and directives, also known as knobs or optimizations. Setting these optimizations to obtain a hardware design with the desired characteristics is a problem that grows exponentially with the number of directives the user adds and complex code structures. The generated hardware is directly associated with the applied directives, but sometimes applying and tuning directives requires considerable effort to obtain a proper hardware implementation. An optimal DSE process grants a hardware design with a good compromise between metrics such as latency, area, throughput, and power consumption.

Over the years, parallel computing models have proven their benefits across different architectures, such as clusters of distributed processors with single cores and multicores, GPU, and cloud. These models act as a bridge between the architecture and the software developer. The actual trend in parallel computer architectures demonstrates progress toward hybrid architectures combining many cores, superscalars, SIMD, hardware accelerators, and on-chip communication systems, which require handling computations and data locality at several levels to achieve suitable performance [15].

Using computing models, methodologies, and frameworks to predict the performance of FPGA/SoC architectures may reduce design times and improve productivity, which are critical issues when choosing these architectures.

Currently, the existing design cycles for FPGA-based reconfigurable architectures are long, owing to the complexity of the architecture and the different applications in multiple research areas. Therefore, for all of the above, a methodology that includes a performance estimation model and exploration of the design space is necessary to analyze the system for its subsequent implementation in hardware.

Applications in the field of image analysis are a relevant research focus in the scientific com-

munity [16–18]. The growth of artificial vision techniques for the processing, recognition, and classification of images has made it possible to expand the expectations of systems to solve problems that are otherwise much more difficult or impossible in different areas, such as security, industry, and autonomous driving.

As presented in [1], in recent years, machine learning (ML) techniques have been applied in multiple fields such as fluid dynamics, high-energy physics, information retrieval, image processing, video processing, security, and biology [19–21]. Because of this trend, models for FPGA-based architectures are being developed to accelerate ML applications with efficient exploitation of hardware resources to improve productivity in the design phase [22–24].

Therefore, to evaluate this thesis proposal, the chosen applications are in the context of image analysis, high-energy physics, and information retrieval.

1.2 Hypothesis

It is possible to develop a methodology for FPGA/SoC architectures composed of analytical models and integrated with a DSE engine based on mathematical programming, guiding the exploration process through HLS rules, to estimate performance metrics, while improving the productivity of the hardware developers.

1.3 Objectives

This thesis aims to efficiently develop, deploy, and evaluate image analysis and highly demanding applications on SoC-based FPGA. Therefore, this research proposes using a methodology, algorithms, and associated software tools for FPGA/SoC hardware acceleration, including a performance estimation model for applications deployed on reconfigurable hardware accelerators based on FPGA.

The estimated metrics include latency and resource utilization. Moreover, a design space explorer engine is incorporated to provide a set of hardware designs that satisfy the design conditions defined by the hardware designer.

1.3.1 Specific objectives

- Exploration, proposal, and development of performance models for FPGA/SoC to estimate mainly latency and resource utilization.
- Investigation, proposal, and development of a methodology for the exploration of the design space for FPGA.
- Implementation of highly demanding applications on FPGA/SoC, including pulse shape discriminator for cosmic rays based on ML, deep neural networks for automatic image classification, and re-ranking for information retrieval.

1.4 Contribution

The contributions of this thesis are as follows:

- MARTE as a comprehensive performance estimator for hardware acceleration, composed of analytical models to estimate area and latency and a DSE engine for providing high-level insights to hardware developers
- MARTE DSE engine, compound of two independent sub-engines:
  - DSE single-objective Bayesian optimization for latency.
  - DSE evolutionary multi-objective optimization for latency-area.
- Integration of MARTE with the Roofline model, the leading parallel computing model adapted for FPGAs.
- An exhaustive evaluation of a methodology to deploy deep neural networks classifier on SoC/FPGA.
- Hardware acceleration of:
  - A pulse shape discriminator for cosmic rays to be paired with a front-end data-acquisition system based on SoC-FPGAs for water Cherenkov detectors (WCD).
  - Automatic pest classification based on convolutional neural networks (CNN).
  - Re-ranking algorithm for information retrieval.

1.5 Methodology

Fig. 1.1 presents the methodology proposed in this thesis to address the gap between applications and FPGA/SoC architectures and, in turn, obtain an efficient hardware design for image analysis and highly demand applications using the HLS tool. Two complementary strategies are considered:

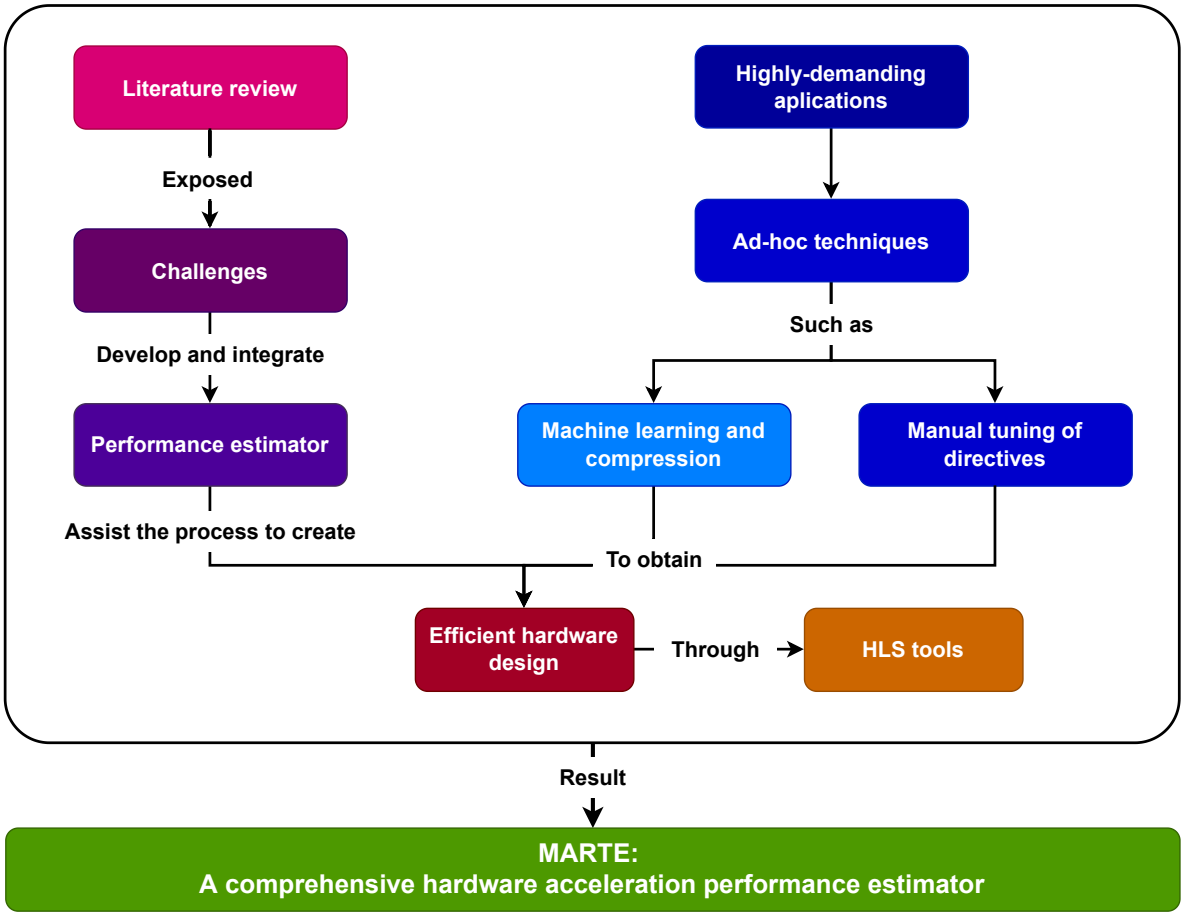


Figure 1.1: Methodology.

- **Performance estimator:** the literature review shows that existing performance estimators based on models, methodologies, and frameworks proposed for FPGA/SoC still need to overcome different challenges to be widely used for estimating system performance in the early stages of design. Moreover, this type of tool could assist the hardware developer in the process of creating efficient hardware designs.

- **Ad-hoc techniques:** an efficient hardware design can be obtained by combining different

techniques to satisfy the user and system constraints. These techniques can be related to the application or tool domains. The former includes the use of methods such as ML, compression, and efficient algorithms to map applications on FPGA/SoC. The latter is based on the options offered by the HLS tool, such as compiler directives, which require considerable effort to obtain a suitable configuration to improve the throughput, latency, and resource utilization.

To overcome the exposed above, MARTE, a comprehensive performance estimator for hardware acceleration is proposed to effectively predict the resource utilization and latency for FPGA/-SoC, building a bridge between the application and architectural domains. Furthermore, image analysis and other highly demanding applications (such as automatic pest classification, pulse shape discriminator for cosmic rays, and re-ranking for information retrieval) are considered in this thesis to achieve the objectives, emphasizing the benefits when this type of application is traversed by compression techniques when targeting FPGA/SoC devices.

Regarding the application object in this thesis, different problems are explored in relevant areas, considering highly demanding applications:

- **Pulse shape discrimination:** data-driven learning based on neural network (NN) models for pulse shape discriminator (PSD) in the context of Water Cherenkov detectors (WCD) used in LAGO (The Latin American Giant Observatory) experiment [25], aiming to perform pulse discrimination in a front-end data-acquisition system based on FPGA/SoC. The PSD identifies four types of pulses (based on the shape) obtained from the corresponding data acquisition system.

- **Image classification:** automatic pest classification based on ML to be deployed in an embedded system, considering two types of classes: insects and the corresponding plague (moth).

- **Re-ranking algorithm:** exploitation of the parallelism on FPGA/SoC devices for the re-ranking algorithm QUICKSCORER (QS), the state of the art algorithm for performing fast inference with tree ensembles [26, 27]. QS exploits a particular representation of the tree ensemble based entirely on linear arrays accessed with high locality. This characteristic permits a very fast traversal of the tree ensemble at inference time.

1.6 Challenges

Nowadays, the explosive growth of accelerators promises greater computational capabilities. FPGA/SoC devices are widely used as hardware accelerators in different areas of research and development. However, there is the necessity to address some challenges, as presented in [1]. Even using HLS tools, reconfiguring an SoC-based FPGA with an efficient hardware design is a challenging task. This is easily made apparent by some observations [1]:

- Physical resources, such as memory bandwidth, reconfigurable hardware (LUTs, CLBs, and slices), and static hardware (DSPs and BRAMs) are limited in FPGA/SoC devices. Thus, the available physical resources should be used skillfully, considering techniques to improve the latency, area, and power.
- Code restructuring techniques aid in creating efficient FPGA implementations using HLS tools, modifying the source code of the application according to the FPGA architecture.
- The number of processing elements (PE) in a hardware design, and consequently the level of coarse-grain parallelism that can be obtained, is limited to the available physical resources. Therefore, different strategies should be implemented to exploit the architecture to increase the system's scalability.
- There is a trade-off between the different metrics to be optimized. For example, the area occupied is likely to increase if the latency is reduced, and vice versa. Thus, the FPGA designer should choose a good compromise between the metrics in terms of resources, computing operations, and throughput, among others.
- The hardware generated through HLS tools is directly associated with the applied directives, but sometimes applying and tuning directives require considerable endeavour to obtain a proper FPGA implementation. Moreover, generating a solution for each directive combination is associated with the synthesis time, reducing productivity.
- The exploration of the design space is linked to the human effort of performing combinations of directives, user design constraints, FPGA features, and code restructuring, among others.

Hardware designers can cope with the above considerations through performance estimators to reduce design time, as follows:

- The coarse-grain parallelism level can be obtained by employing a Roofline model, identifying the computation-to-communication ratio, and exposing the relationship between communication bottlenecks, computations, and the number of replicas.
- Design space explorers aim to identify the optimal combination of directives to obtain an HLS-based hardware design with a proper trade-off among different metrics, generating the Pareto-optimal set of designs. Reducing the design space and avoiding HLS in the exploration process can improve the design time.
- Models integrated within a methodology or framework can automatically estimate the performance of HLS-based hardware designs without executing HLS tools.
- Some frameworks and methodologies including DSE provide automatic directive-insertion optimizations and code transformation insights.

Nevertheless, the literature review shows that a number of challenges has to still be addressed in order to make optimal use of performance estimators (models, methodologies, and frameworks) such as:

- Recent HLS tools generate more comprehensive reports with more accurate information on total resource availability, latency, clock frequency, and resource utilization. These reports can be integrated with models, methodologies, and frameworks to estimate metrics and provide an initial value for the replication factor of a single PE. However, the report generation is linked to the synthesis time of the FPGA implementation. Reducing the design time is essential when using FPGA/SoC without losing hardware quality to reconfigure the platform. Thus, if the HLS tool is in the loop for performance estimation using reports, it can lead to an increased design time.
- The performance metrics reported by HLS tools make them suitable to be combined with a parallel computation model to reduce the time required to obtain the necessary statistics for each implementation for a specific application. However, there is a gap between the HLS report and the real hardware implementation that can be addressed with a performance model that includes the results obtained from the sourceCode-to-bitstream flow using the values related to final hardware utilization, power consumption, and timing reports.

- Computing models for FPGA-based reconfigurable hardware accelerators have to consider that the inherent hardware is not fixed. Rather, it is defined by how the application is described. Therefore, a higher number of parameters have to be included in the model, such as hardware resources (DSP, BRAM, LUT, and FF), programmable logic clock, latency, byte-operations (Bops), scalability in the number of PE, and power consumption. This contrasts with the computing models proposed for other parallel platforms, such as PRAM or BSP, that use a few parameters. Nevertheless, including more parameters in the model increases the analysis accuracy but affects the complexity of the model analysis. Therefore, the trade-off between these two features has to be addressed. In addition, the parameters should be adjusted according to the particular combination of directives applied to the source code.

- The compatibility among different versions of HLS tools is not granted by models, methodologies, and frameworks. As a consequence, calibration techniques can help maintain compatibility between high-level tools, thereby avoiding being tied to one version of HLS tool in particular [28].

Moreover, when a DSE engine is integrated with models, methodologies, and frameworks, the following aspects need to be considered:

- One of the key points in the DSE is the execution of HLS tools during the exploration stage to validate the configuration obtained. This behaviour can lead to a long runtime, becoming a drawback in the DSE phase. Therefore, the adoption of different techniques to reduce the execution time of the exploration phase is indispensable.

- It is often sufficient to find a suboptimal combination of knobs based on specific metrics and user constraints. An important strategy is pruning the design space using intermediate Pareto-optimal designs, giving priority to the points that permit high-performance behaviours.

- The DSE engine should guarantee a good compromise among the QoR and performance metrics.

- Approximate computing [29] can lead to an expansion of the design space, generating Pareto-optimal designs with a trade-off between area-power-latency estimation and error compu-

tation. A reduction in the space to be explored is fundamental to minimizing the invocations of HLS tools.

- It is important to identify the strengths and weaknesses of a given design space explorer. This can be performed using benchmarks.
- Mapping an optimal design from the DSE to the FPGA/SoC can be challenging while maintaining the QoR reported by the DSE engine, mainly latency. In the process of mapping the final hardware design onto the FPGA/SoC, the place-and-route phase plays an important role and different strategies provided by commercial tools can be used in this phase, adding another factor to be analyzed.
- It is fundamental to consider the application of HLS-specific compiler optimizations, due to the impact that they have on the hardware quality, in terms of latency, area, and power consumption [30].

1.7 Scientific publications

This research has generated the following original scientific contributions:

Thesis related

- **Molina, R.S.**, Gil-Costa, V., Crespo, M. L, Ramponi, G. (2022) "High-Level Synthesis Hardware Design for FPGA-based Accelerators: Models, Methodologies, and Frameworks". In IEEE Access, vol. 10, pp. 90429-90455, 2022. IEEE.
- **Molina, R. S.**, Carrer. V., Ballina; M., Crespo, M. L., Bollati, L., Sequeiro, D., Marsi, S. Ramponi, G. (2022) “ML-based classifier for precision agriculture on embedded systems”. In International Conference on Applications in Electronics Pervading Industry, Environment and Society [Accepted].
- Gil-Costa, V., Loor, F., **Molina, R.S.**, Nardini, F. M., Perego, R., Trani, S. (2022) “Energy-Efficient Ranking on FPGAs through Ensemble Model Compression”. In 12th Italian Information Retrieval Workshop.

- **Molina, R. S.**, Garcia, L. G., Morales, I. R., Crespo, M. L., Ramponi, G., Carrato, S., Cicuttin, A., Perez, H. (2022). "Compression of NN-Based Pulse-Shape Discriminators in Front-End Electronics for Particle Detection". In International Conference on Applications in Electronics Pervading Industry, Environment and Society, pp. 93-99. Springer, Cham.

- Gil-Costa, V., Loor, F., **Molina, R. S.**, Nardini, F. M., Perego, R., Trani, S. (2022). "Ensemble Model Compression for Fast and Energy-Efficient Ranking on FPGAs". In European Conference on Information Retrieval, pp. 260-273. Springer, Cham.

- Suárez, A., **Molina, R. S.**, Ramponi, G., Petrino, R., Bollati, L., Sequeiros, D. (2021, November). "Pest detection and classification to reduce pesticide use in fruit crops based on deep neural networks and image processing". In 2021 XIX Workshop on Information Processing and Control (RPIC), pp. 1-6. IEEE.

- **Molina, R. S.**, Loor, F., Gil-Costa, V., Nardini, F. M., Perego, R., Trani, S. (2021). "Efficient traversal of decision tree ensembles with FPGAs". In Journal of Parallel and Distributed Computing, 155, 38-49. Elsevier.

- García Ordóñez, L. G., **Molina, R. S.**, Morales Argueta, I. R., Crespo, M. L., Cicuttin, A., Carrato, S., Ramponi, G., Pérez Figueroa, H. E., Ballina Escobar, M. G. (2021). "Pulse shape Discrimination for Online Data Acquisition in Water Cherenkov Detectors Based on FPGA/SoC". In 37th International Cosmic Ray Conference (ICRC2021), p. 274. PoS Sissa.

- Marsi, S., Bhattacharya, J., **Molina, R. S.**, Ramponi, G. (2021). "A Non-Linear Convolution Network for Image Processing". In Electronics 2021, 10, 201. MDPI.

- Garcia, L. G., **Molina, R. S.**, Crespo, M. L., Carrato, S., Ramponi, G., Cicuttin, A., Morales, I. R. Perez, H. (2021). "Muon–Electron Pulse Shape Discrimination for Water Cherenkov Detectors Based on FPGA/SoC". Electronics 2021, 10, 224. MDPI.

- Guzzi, F., De Bortoli, L., **Molina, R. S.**, Marsi, S., Carrato, S., Ramponi, G. (2020). "Distillation of an End-to-End Oracle for Face Verification and Recognition Sensors". In Sensors, 20(5), 1369. MDPI.

Others

- Cicuttin, A., Morales, I. R., Crespo, M. L., Carrato, S., García, L. G., **Molina, R. S.**, Valinoti, B., Folla Kamdem, J. (2022). "A Simplified Correlation Index for Fast Real-Time Pulse Shape Recognition". In Sensors, 22(20), 7697. MDPI.

- Florian Samayoa, W., Valinoti, B., **Molina, R. S.**, Garcia Ordonez, L. G., Crespo, M. L., Carrato, S., Cicuttin, A., Levorato, S. (2022). "Diagnostic analytics for pixelated particle detectors: A case study". In International Conference on Applications in Electronics Pervading Industry, Environment and Society [Accepted].

- Crespo, M. L., Foulon, F., Cicuttin, A., Bogovac, M., Onime, C., Sisterna, C., Melo, R., Florian Samayoa, W., García Ordóñez, L. G., **Molina, R. S.**, Valinoti, B. (2021). Remote Laboratory for E-Learning of Systems on Chip and Their Applications to Nuclear and Scientific Instrumentation. In Electronics, 10(18), 2191. MDPI.

- Guillermo, G. L., Crespo, M. L., Carrato, S., Cicuttin, A., Oswaldo, F. W., **Molina, R. S.**, Valinoti, B. Levorato, S. (2021). "High Voltage Isolated Bidirectional Network Interface for SoC-FPGA Based Devices: A Case Study: Application to Micro-pattern Gaseous Detectors".In Applications in Electronics Pervading Industry, Environment and Society. Lecture Notes in Electrical Engineering, vol 738, pp. 280-285. Springer, Cham.

- **Molina, R. S.**., Gonzalez, V., Benito, J., Marsi, S., Ramponi, G., Petrino, R. (2021). "Implementation of Particle Image Velocimetry for Silo Discharge and Food Industry Seeds". In Applications in Electronics Pervading Industry, Environment and Society. Lecture Notes in Electrical Engineering, vol 738, pp. 3-11. Springer, Cham.

Awards

- **Industry impact award:** Gil-Costa, V., Loor, F., **Molina, R. S.**, Nardini, F. M., Perego, R., Trani, S. (2022). "Ensemble Model Compression for Fast and Energy-Efficient Ranking on FPGAs". In European Conference on Information Retrieval, pp. 260-273. Springer, Cham.

1.8 Thesis outline

The thesis is composed of nine main chapters, introduced as follows:

- Chapter 2 discusses specific topics in ML and mathematical programming. Because the applications focused on in this thesis are based on ML and compression techniques for their implementation in hardware, the basic concepts, ML-based model compression techniques, and methodology to deploy deep neural network classifiers on FPGA/SoC are introduced. Mathematical programming concepts for single- and multi-objective optimizations, which are used for DSE engine implementation, are introduced.

- Chapter 3 describes the background of FPGA/SoC and parallel computing models for performance estimation. FPGA/SoC architecture is presented with DSE, the main metrics for this technology, and the techniques to improve latency, area, and delay. As a model for performance estimation is proposed in this thesis, the leading parallel computing models for performance estimation are introduced.

- Chapter 4 summarizes and discuss the state of the art in performance estimators for FPGA-based reconfigurable hardware accelerators, providing a classification in general approaches and DSE. Moreover, their use for highly-demanding applications is exposed.

- Chapter 5 introduces MARTE, a comprehensive performance estimator for hardware acceleration through FPGA/SoC, composed by analytical models and a DSE engine based on single- and multi-objective optimizations. The different models are introduced as the techniques employed for their implementation.

- Chapter 6 presents the different cases of study and their corresponding hardware acceleration: pulse shape discriminator for cosmic rays, automatic pest classification, and re-ranking algorithm for information retrieval. The applications are traversed for an ensemble of compression techniques, to obtain a suitable implementation on FPGA/SoC.

- Chapter 7 discusses the performance evaluation of MARTE. The first stage of the experiments was aimed at evaluating MARTE with different applications (basic and highly demanding). In the second stage, the assessment of MARTE DSE engine is exposed, showing the efficiency and effectiveness of the engine to provide high-performance configurations.

- Chapter 8 provides a way to pair MARTE with the Roofline model, the leading parallel computing model adapted to FPGA architectures.
- Chapter 9 presents conclusions and future directions.

Chapter 2

Specific topics in machine learning and mathematical optimization

This chapter presents specific topics in machine learning (ML) and mathematical programming. Because the applications focused on in this thesis are based on ML and compression techniques for their implementation in hardware, Section 2.1 introduces the basic concepts, ML-based model compression techniques, and a methodology to compress ML architectures efficiently. Section 2.4 presents the concepts associated with mathematical programming for single- and multi-objective optimizations used for the DSE engine development and implementation.

2.1 Deep neural networks

Machine learning is a subfield of artificial intelligence (AI), and it aims to build analytical models based on data capable of learning and to improve performance [31]. Based on the learning process, ML techniques can be classified as supervised, semi-supervised, and unsupervised [32].

- **Supervised learning:** the algorithms are pre-trained on a fully labelled dataset. The main objective of this type of learning is predicting results from an input. In turn, classification and regression models can be found where the output of the function can be a numerical value (as in regression problems) or class label (as in classification problems).
- **Semi-supervised learning:** labelled (the least quantity) and unlabelled (the largest quantity) data are used during the training phase.

- **Unsupervised learning:** these algorithms do not have prior knowledge and aim to find patterns in the dataset, making it possible to organize them in some way. It is self-organized; there are only input data without labels and a cost function to be minimized.

In supervised learning, an artificial neural network (ANN) [33] is composed of neuron (or node) interconnections arranged in different layers, usually an input layer, middle or hidden layers, and an output layer, where a prediction is generated. The connections between the neuron and inputs (x_i) are called weights (w_i). Each node has several inputs and only one output, and the ANN uses a nonlinear activation function to compute the output value. This description is shown in Fig. 2.1 and is mathematically represented by Eq. (2.1).

$$y = f((\sum_i x_i w_i) + b) \tag{2.1}$$

where x_i are the inputs, w_i are the weights, b represents the bias, f the activation function and y the final output of the neuron.

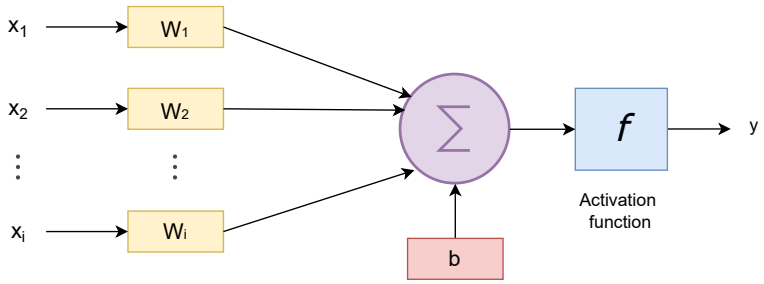


Figure 2.1: Detail of a single neuron function.

In an ANN-based classifier, the input is mapped to a specific class. For this task, an ANN goes through a supervised training step to recognize patterns and compares its actual output with the desired output. The difference between these two values is adjusted with backpropagation.

Convolutional neural networks (CNN) are a type of deep neural network (DNN) that receives an image as input and extracts features directly from it, learning as the network trains on a collection of images. The architecture is defined through a stack of layers, and each layer transforms one volume of activation into another through a differentiable function. CNNs are mainly made up of two stages: feature extraction (composed of tens or hundreds of hidden layers based, mainly, on convolutional and pooling layers) and classification (based on fully-connected layers) [32]. For

more details in this topic, the reader can refer to [32].

DNNs are mainly composed of parameters and hyperparameters; the former are defined during training. The user defines the latter before training, characterizing the architecture by defining the number of layers and filters and the learning approach, among others. Techniques such as **random search** [34], **random grid** [35], and **Bayesian optimization** (BO) [36, 37] are mainly used for tuning the hyperparameters of DNNs, thereby avoiding the trial-error procedure for their selection [34,38,39]. Random search selects random trials and is a greedy approach that settles for local optimality, failing to end with global optimality. For a given hyperparameter space, grid search tries all the possible combinations of hyperparameters. BO is a sequential approach that uses the information from previous steps, scaling with the utmost resource utilization, handling noisy data well, and exploiting non-continuous spaces to attain global minima [40].

2.2 Compression for ML-based models

Compression techniques are essential for deploying machine learning models on resource-constrained devices while maintaining efficiency and effectiveness and obtaining smaller and faster models [41, 42].

Among the most commonly used techniques for compression are **pruning**, **quantization** [43], and **knowledge distillation** [44, 45], which can be combined to benefit the compression process with their unique characteristics. Pruning and quantization [42, 46] are orthogonal to distillation, helping to achieve better performance by reducing the size of the model with minimum loss of accuracy [47]. Pruning aims to reduce the number of parameters by removing neurons and connections, whereas quantization reduces the memory footprint by selecting the number of bits representing weights and biases.

Quantization-aware training (QAT) [42,48] and quantization-aware pruning (QAP) [49] are learning processes based on quantization methods. The former avoids post-training quantization, and the quantization operations on weights and activations are simulated while maintaining floating-point precision to update the weights and compute the gradient. The latter performs a QAT by integrating pruning.

Finally, knowledge distillation (KD) [45] is devoted to transferring the knowledge (or "dark knowledge" according to Hinton) from a teacher network (a single large model or an ensemble

of models) to a smaller and faster target network (distilled or student) that can mimic the teacher's behavior, being computationally less expensive. Fig. 2.2 presents the process involved in KD. a

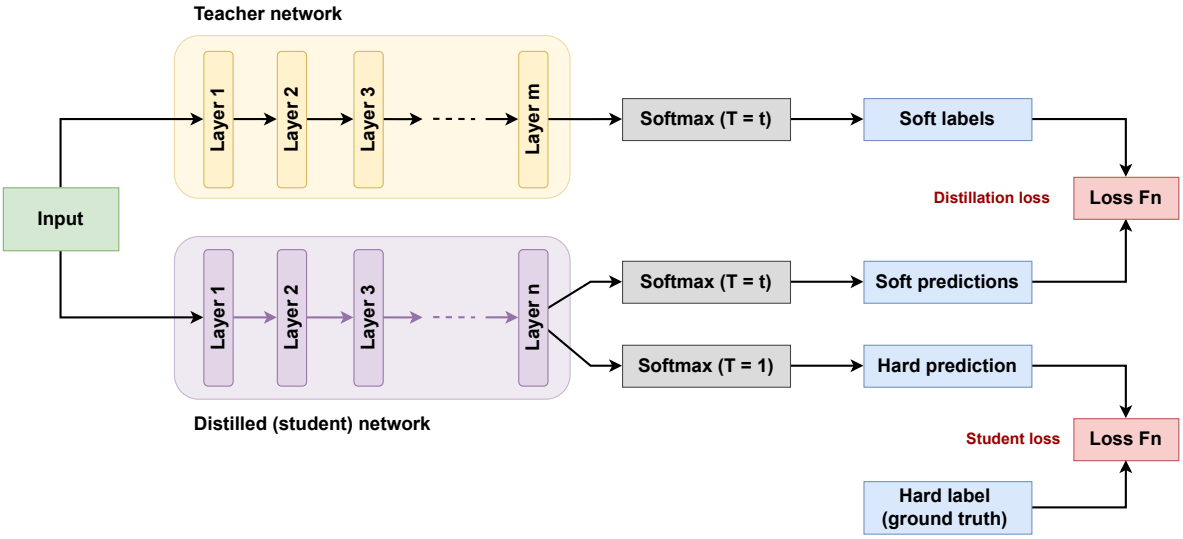


Figure 2.2: Knowledge distillation process.

In KD, the probability q_i of class i is calculated from the logits z as presented in Eq. 2.2, where T represents the temperature. The probability distribution obtained by the Softmax function becomes softer when T grows, providing more information, called by Hinton the "dark knowledge". If $T = 1$, the result is the same as the Softmax activation function [45].

$$q_i = \frac{e^{(\frac{z_i}{T})}}{\sum_j e^{(\frac{z_j}{T})}} \tag{2.2}$$

2.3 Methodology to deploy DNN-based classifiers on SoC

As the final goal is the implementation of an ML-based classifier in the FPGa/SoC platform, model compression is performed using an ensemble of different techniques: quantization, pruning, and KD. Fig. 2.3 shows the steps involved in the methodology used to obtain efficient compressed neural network-based classification models, supported by the literature in contributions such as [50, 51].

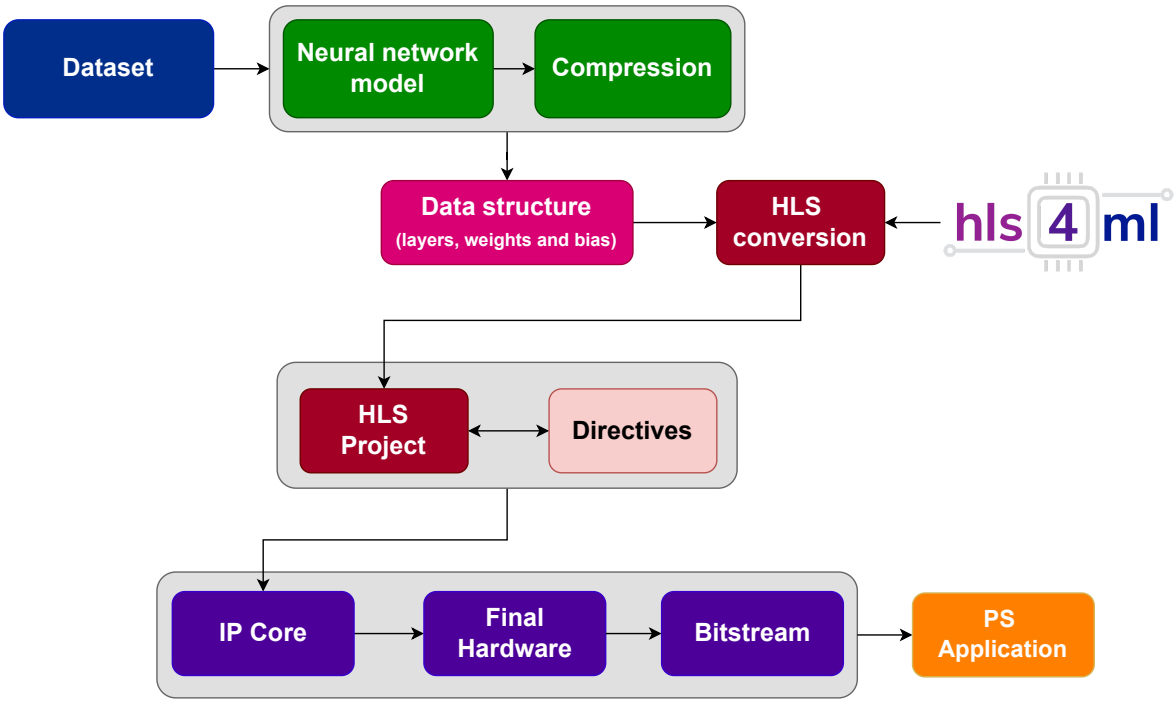


Figure 2.3: Methodology. The input is the labeled dataset used to train the teacher and student (or target) networks. After this, the data structure generated is the input for the hls4ml [52] package, translating the neural network-based model into a HLS project.

The input of the methodology is the labeled dataset used to train the target network. Then, the data structure generated is the input for the hls4ml package [52], translating the neural network-based model into an HLS project. Once the IP core is generated, it can be integrated with the final system (hardware and software).

Fig. 2.4 presents the training and compression stages integrated within the methodology presented in Fig. 2.3 (green boxes) to generate the final architecture to be deployed on SoC-based FPGA. Each stage is detailed as follows:

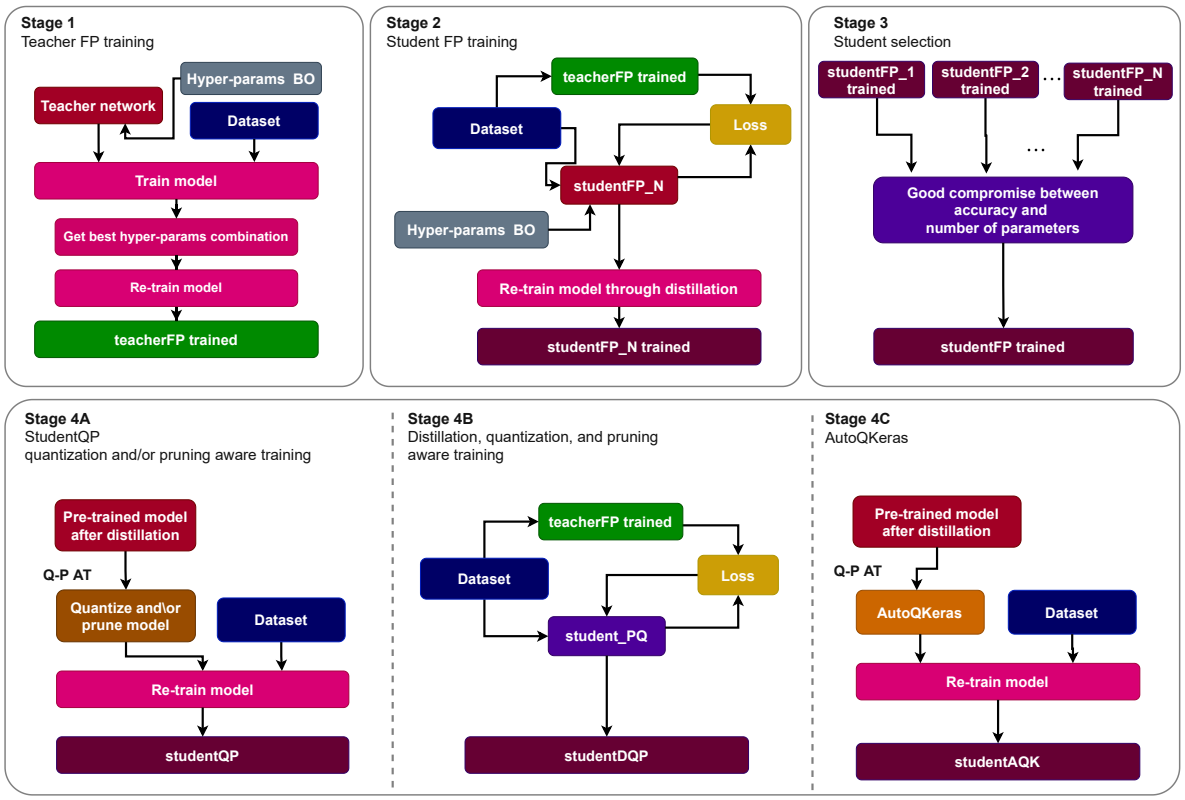


Figure 2.4: Compression workflow to deploy DN classifiers on FPGA/SoC.

- **Stage 1 - Teacher floating-point (teacherFP) training:** The teacher network is defined employing Bayesian optimization (BO) for hyperparameters tuning. Given an architecture, the algorithm searches for the best configuration to optimize a specific objective function, such as accuracy, by modifying the number of neurons, kernels, and learning rate. After tuning, the network is trained with the best configuration, generating a teacher model with 32-bit floating-point precision.

- **Stage 2 - Distillation. Student floating-point (studentFP) training:** In this step, the knowledge of the teacherFP is distilled into the student architecture. The hyperparameters for the student network are chosen by BO, similar to [53]. Several studentFP networks can be trained by varying the number of layers and other hyperparameters.

- **Stage 3 - Student selection:** The studentFP to be accelerated in hardware is selected, considering a good compromise between accuracy and the number of parameters.

- **Stage 4 - Student quantization- and pruning-aware training:** This step aims to obtain the quantized student (studentQ), that can be generated through different stages:

  - **Stage 4A:** From the pre-trained model after distillation, a quantization-aware or ebpruning-aware training is performed to generate the compressed student (studentQP).
  - **Stage 4B:** The knowledge of the teacherFP is distilled to a student network (student-DQP), previously defined with quantization and pruning strategies.
  - **Stage 4C:** From the pre-trained model after distillation, the number of bits for each layer of the student is chosen through BO, generating the compressed network (studentAQP). This approach can optimize the network based on energy and compression ratio goals.

2.4 Mathematical optimization

Mathematical optimization is a field of study that comprises problems in which it is desirable to maximize or minimize a real function defined on the objective space $\mathbb{R}^n$, and whose variables belong to a predefined set [54].

A **single-objective optimization problem** can be defined by Eq. 2.3, considering $f(\mathbf{x})$ the objective function to be optimized and $\mathbf{x}$ a p-dimensional decision variable vector $f(\mathbf{x}) = (x_1, ..., x_p)$ [55], thus $\mathbf{x} \epsilon \Omega$, with Ω as the feasible region [54, 56].

$$\min_{\mathbf{x}} f(\mathbf{x}) \tag{2.3}$$

The objective function $f(\mathbf{x})$ is subjected to constraint functions of the problem presented in Eq. 2.4, which can be expressed by inequality ($g_i(\mathbf{x})$) and (or) equality ($h_i(\mathbf{x})$), where m, n represent the total amount of constraints for each case.

$$\begin{aligned} g_i(\mathbf{x}) \leq 0 \quad i = 1, 2, ...m \\ h_i(\mathbf{x}) = 0 \quad i = 1, 2, ...n \end{aligned} \tag{2.4}$$

A **multi-objective optimization problem** can be defined as minimizing or maximizing $F(\mathbf{x}) = (f(\mathbf{x}_1), f(\mathbf{x}_2), ..., f(\mathbf{x}_k))$, with k objective functions. $\mathbf{x}$ is a p-dimensional decision variable vector

$f(\mathbf{x}) = (x_1, ..., x_p)$, thus $\mathbf{x} \epsilon \Omega$, . The objective function $F(\mathbf{x})$ is subject to $g_i(\mathbf{x}) \leq 0, i = 1, 2, ...m$ and $h_i(\mathbf{x}) = 0, i = 1, 2, ...n$, which are the constraints of the problem [54, 56].

2.4.1 Bayesian optimization

Black-box optimization technique (BBO) [57] refers to a problem in which an objective function should be optimized (maximize or minimize) through a black-box interface. The optimization algorithm may query the value $f(x)$ for a point x, but it does not obtain gradient information. In particular, it cannot make any assumptions on the analytic form of f.

BBO, based on Bayes' theorem, aims to find the minimum or maximum of an objective function. The conditional probability of an event is given by the Eq. 2.5:

$$P(A|B) = P(B|A) * P(A)/P(B) \tag{2.5}$$

Removing the normalization faction $P(B)$, then $P(A|B)$ is defined as in Eq. 2.6, where $P(B|A)$ is known as the likelihood probability and $P(A)$ the prior probability.

$$P(A|B) = P(B|A) * P(A) \tag{2.6}$$

Considering x_i as a set of samples of the exploration space, and $f(x_i)$ the objective function to be evaluated in x_i, the available data D is composed by the sequential collection of the samples and their outcomes, then $D = \{x_i, f(x_i), ..., x_n, f(x_n)\}$, and is used to obtain the prior probability. The function $P(D|f)$ presented in Eq. 2.7 represents the probability of observing the data given the function f and is defined as the likelihood function.

$$P(f|D) = P(D|f) * P(f) \tag{2.7}$$

The **surrogate function** (posterior probability), usually defined as a Gaussian process (GP), is the Bayesian approximation of the objective function that can be sampled efficiently, identifying the points that are promising minima and updating the surrogate function accordingly. The **acquisition function** ($a(x)$) is the technique by which the posterior is used to select the following sample from the search space, thus, updating the surrogate function with a good compromise between exploitation and exploration [58, 59]. Among the most used acquisition functions are expected

improvement (EI), probability of improvement (PI), and GP – LCB [58, 60].

For further information, the reader can refer to a survey presented in [61].

2.4.2 Multi-objective optimization based on evolutionary algorithms

Evolutionary algorithms (EAs) are heuristic-based approaches inspired by living organisms, emulating their behavior to solve problems [62]. The leading operators associated with EAs are mutation, recombination (crossover), and selection, which help to guide the search towards the solutions.

Within EAs, genetic algorithms (GA) are inspired by Charles Darwin's theory of natural evolution. In GA, a population is composed of individuals represented by a chromosome encoded in a string. The encoding technique may be binary, permutation, tree, or value, among others. Crossover operation selects genes from parent chromosomes and creates a new offspring, while mutation randomly changes the genes of the new offspring. Selection means that the better chromosomes pass their genes to the next generation of the algorithm, according to a fitness criterion.

In evolutionary algorithms, elitism implies that they keep the best solutions, called elitist, of the previous iterations and insert them into the next generation, speeding up the algorithm's convergence [63, 64]. Among the elitist multi-objective evolutionary algorithms (MOEAs), the elitist non-dominated sorting genetic algorithm (NSGA-II) [65–67] is one of the most used MOEAs in real-world applications. It is used in different research areas such as communications, control systems, image recognition, manufacturing, and traffic engineering [68, 69]. The NSGA-II uses an elitist principle, emphasizes non-dominated solutions, and uses crowding distance as an explicit diversity-preserving mechanism. The complexity of NSGA-II is at most $O(MN^2)$, considering N as the population size and M as the number of objective functions [69]. In addition, it has non-penalty constraint handling and fast and efficient convergence.

A general flow diagram of the NSGA-II is presented in Fig. 2.5. The process starts with creating a random population of randomly mutated solutions and crossover with each other to generate new solutions. They are then downsampled back to the initial population size so that the preferred solutions are elected to remain (and survive the evolution). This process is repeated until the convergence criterion is satisfied, generating the final output [70].

For additional information on this topic, the reader can refer to [54, 69, 70].

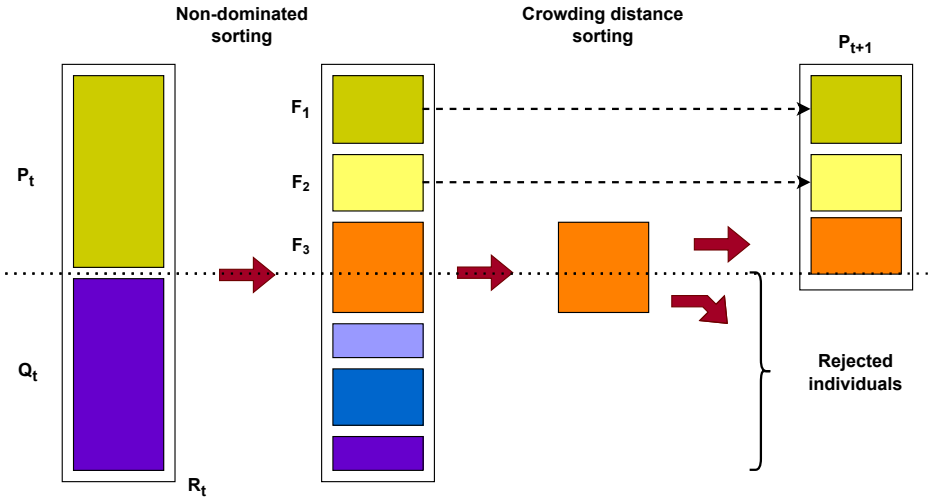


Figure 2.5: General flow diagram of the NSGA-II algorithm. Based on [70].

Chapter 3

Background on SoC-based FPGA and parallel models

This chapter presents the theoretical background associated with the topics addressed in this thesis. Section 3.1 describes the SoC-based FPGA architecture, design space exploration, the primary metrics for this technology, and the techniques to improve latency, area, and delay. As a performance estimation model is proposed in this research, Section 3.2 introduces the leading parallel computing models for performance estimation. Finally, Section 3.3 presents a summary.

3.1 SoC-based FPGA

FPGA architectures contain many reconfigurable circuits, which makes them feasible for accelerating applications that require high parallelism, high performance, and low power consumption.

FPGAs have been commonly used with "soft" processors, designed using programmable logic resources instead of being built into the silicon. Because the use of reconfigurable devices has grown in increasingly sophisticated applications, the need for FPGA-based systems, including processors, has arisen.

Integrating a processor and FPGA into a single chip allows the exploitation of different but

This chapter is based on the work published in [1]: **R. S. Molina**, V. Gil-Costa, M. L. Crespo and G. Ramponi, "High-Level Synthesis Hardware Design for FPGA-Based Accelerators: Models, Methodologies, and Frameworks," in IEEE Access, vol. 10, pp. 90429-90455, 2022, doi: 10.1109/ACCESS.2022.3201107.

complementary computational resources presented in both devices. Dumping critical functions to the FPGA while maintaining the data transfer quickly and coherently between the devices helps achieve a system performance boost.

The SoC-based FPGA architecture combines a processing system with programmable logic (FPGA). The architecture also includes specific interfaces that provide high bandwidth and low latency in the connections between the two parts of the SoC-based FPGA device. The processing system has a fixed architecture composed of a "hard" processor and RAM, while the FPGA is entirely flexible for hardware design.

Within this context, a processing element (PE) can perform an entire computation containing all the elements required for its replication, improving the entire system's performance through coarse-grain parallelism. As an example of this architecture, Fig. 3.1 depicts the different components of the Zynq-7000 SoC and Zynq UltraScale+ multiprocessor system on chip (MPSoC) architectures from AMD-Xilinx. We refer to Xilinx because it is one of the leading providers of this technology. Zynq-7000 SoC combines a dual processor with an FPGA. Zynq UltraScale+ MPSoC devices include quad-core and dual-core real-time processors, GPU, and FPGA.

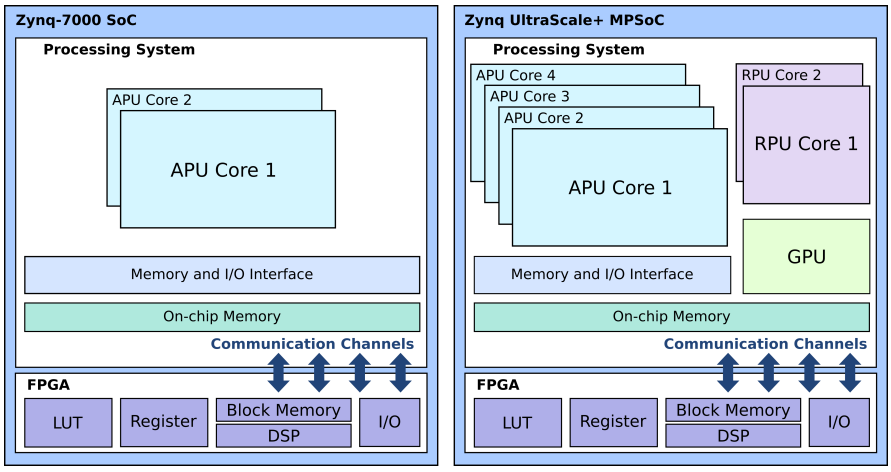


Figure 3.1: Architectures for Zynq-7000 SoC and Zynq UltraScale+ MPSoc devices [1].

3.1.1 Design space exploration and metrics

HLS tools are used to create RTL components from a high-level abstraction using directives to optimize a hardware design described in a high-level language. Each hardware obtained is unique

based on the strategies and optimizations used to describe it. DSE involves the evaluation of multiple implementations with different combinations of directives, also known as knobs or optimizations. In this context, DSE plays an essential role as a fundamental key point in obtaining a hardware design with a good compromise between different metrics.

In the last few years, most DSE techniques have applied multi-objective optimization algorithms (MOOA) dedicated to optimizing objective functions despite conflicting metrics. In this scenario, trade-off solutions form an objective space plotted with the objective values, which builds a Pareto-optimal frontier (PF) and a set of configurations (trade-off solutions) called Pareto-optimal designs.

Let us denote D as the design space composed by q design points, thus $q \epsilon D$. PF can be defined as a set of hardware designs $PF = \{d_1, d_2, \ldots, d_k\}$, where the sub-index k defines the number of elements in PF. Each d_i with $1 \leq i \leq q$ represents a hardware design with unique features such as latency, resource utilization, and clock frequency. Suppose the area (A) and latency (L) are the objective functions. In that case, any hardware design d_i is considered a Pareto-optimal design, and in consequence, $d_i \epsilon PF$, if there is no other design d_n with $1 \leq n \leq q$ in the search space such that it simultaneously has less area (A) and less latency (L) than d_i [28], as shown in Eq. 3.1.

$$A(d_i) \leq A(d_n) \; and \; L(d_i) \leq L(d_n) \tag{3.1}$$

A survey on MOOA for HLS, presented by Fernandez de Bulnes *et al.* [71], remarks on the expansion of these techniques for the FPGA DSE process. The authors conclude that the most common objective functions are: latency (clock cycles), area (LUT, BRAM, DSP, and FF), power (static and dynamic), wire length, digital noise, reliability, temperature, and security. For FPGA-based devices, all metrics should be minimized except reliability and security. The authors remark on six main multi-objective methods applied for HLS DSE: evolutionary algorithms, single-solution-based heuristics, problem-specific heuristics, branch-and-X, learning-based methods, and swarm intelligence systems. Some examples are the studies presented in [72–81].

An overview of the general DSE process using HLS tools in the loop, based on [28], is shown in Fig. 3.2. An application, described mainly in C/C++, SystemC, or OpenCL, is the input of this type of system. A low-level virtual machine intermediate representation (LLVM IR) [82] is obtained from the input code through the Clang front-end compiler [83], generating a control data flow graph

(CDFG). Each node of the graph represents the operations connected by control dependency and data. The DSE phase generates a unique batch of directives to minimize a specific cost function. The HLS tool then uses the generated optimizations, application, and technology library to generate the final optimized RTL.

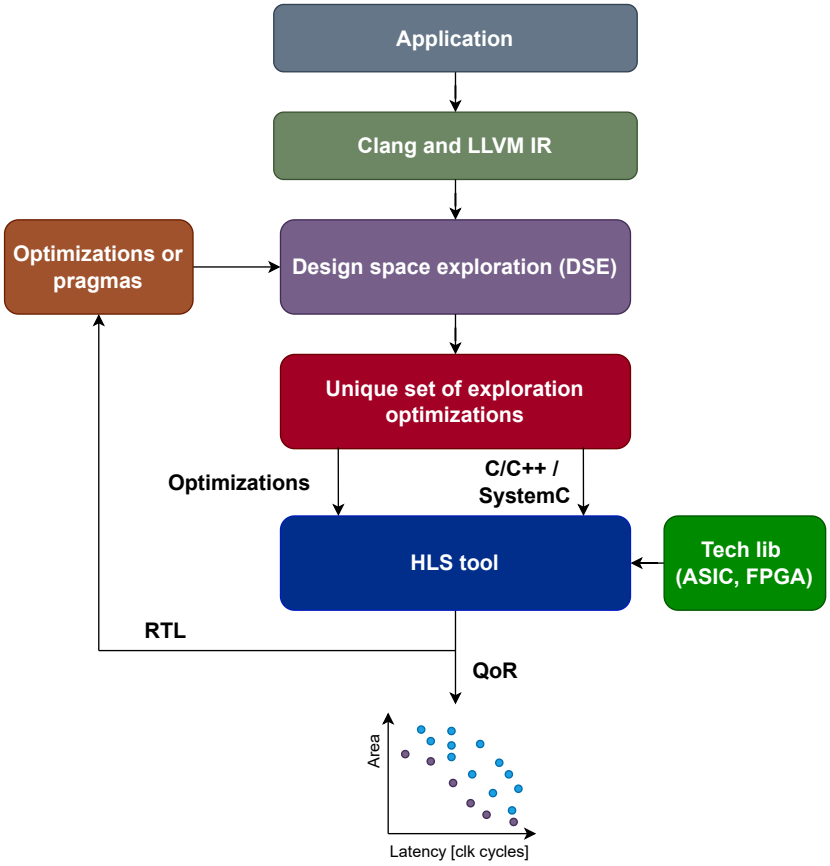


Figure 3.2: Typical DSE framework with HLS in the loop, based on [28]. From [1].

3.1.2 Improving performance with HLS tool

There are different techniques to improve the performance of algorithms running on FPGAs through HLS tools [84]. One of the most common approaches is to **use a set of directives** (or knobs) provided by HLS tools to improve throughput, latency, and resource utilization. To this end, HLS tools insert pragmas (compiler directives) into the source code [7, 9], and some of the most used optimization techniques are:

- **Loops handling:**
  - **Loop inlining:** this directive removes function hierarchy. Nevertheless, a high factor of inlining can create a considerable amount of logic and slow runtime.
  - **Loop merging and flattening:** help to remove the redundant computation among multiple (related) loops. The former merges consecutive loops to increase sharing, improve logic optimization, and reduce overall latency. The latter allows the collapse of nested loops into a single loop with improved latency.
  - **Loop Pipelining:** in the presence of sequential operations executed multiple times, this technique allows the insertion of registers at the output of each stage. Each operation can run in parallel on different input data, increasing the overall throughput at the expense of area. Pipelining can be applied at instruction and function levels.
  - **Loop unrolling:** let us denote f as the unroll factor. One iteration takes n clock cycles in a rolled loop. Thus, f iterations can be executed within n clock cycles when unrolling the loop by a factor of f, and the total latency for the unrolled loop is n/f (without data dependency). This technique can improve both latency and throughput, but it is expensive in resource utilization since it is affected proportionally by f.
- **Memory optimizations:**
  - **Array partition:** let us denote p_f as the partitioning factor. Array partition splits an array in p_f sections to be mapped into a dedicated memory element, allowing multiple simultaneous accesses to it at the cost of higher utilization of memory elements.
  - **Array reshape:** this technique allows creating smaller arrays from the original array, concatenating elements by increasing bit widths, thus reducing the number of BRAM consumed and allowing parallel access to the data.

Fig. 3.3 presents an example of the latency of a given function and its improvement with pipelining and unrolling techniques. For an array, Fig. 3.4 shows the effect of applying memory optimization techniques based on array partition directive.

Nevertheless, memory performance could be affected by array partition techniques because improper partitioning leads to generating numerous multiplexers, incurring additional delays [85].

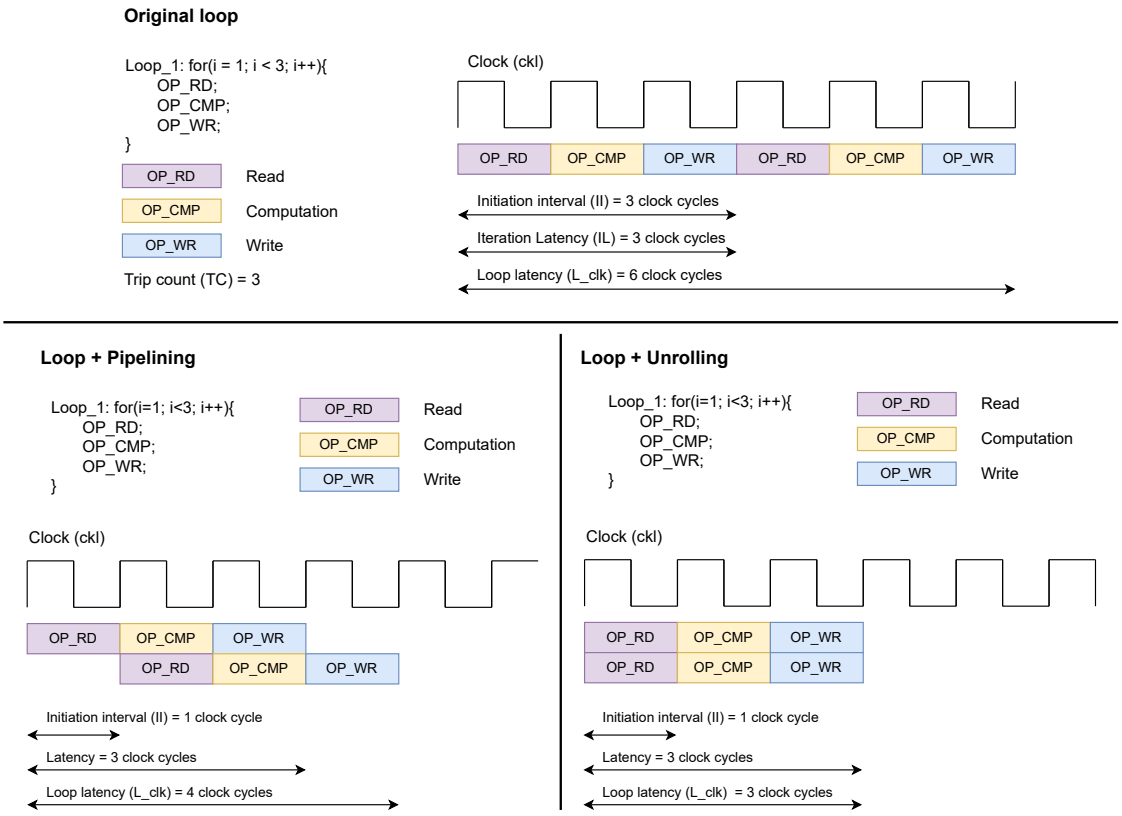


Figure 3.3: HLS directives for loop handling.

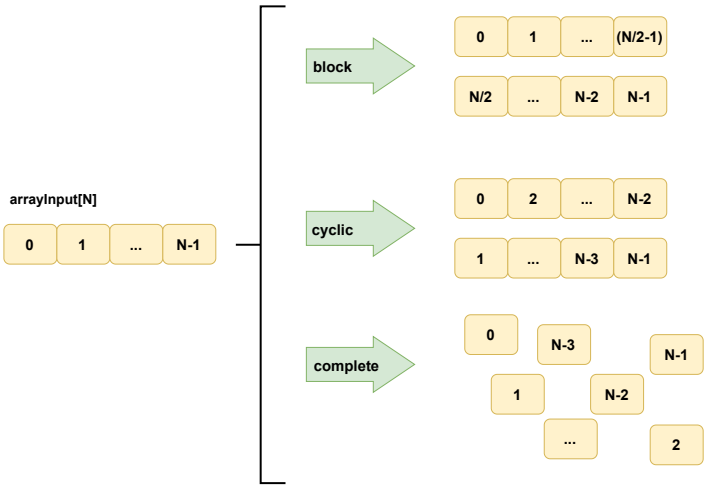


Figure 3.4: HLS directives for memory optimization.

Code restructuring techniques [86–90] are also used to improve the hardware design of the

algorithms. Ferreira *et al.* [91] introduce an approach for automatic code restructuring targeting HLS tools. A detailed survey is presented in [92], where the sets of optimizing transformations techniques are classified into: pipelining, scaling, and memory-enhancing transformations.

Quantization techniques aim to reduce memory footprint by selecting the number of bits representing the data structures and operations to improve objective functions such as latency, resource utilization, and throughput. Moreover, by reducing the computational intensity, the power consumption also decreases [93–96].

3.2 Parallel computing models for performance estimation

Computing models allow to easily analyzing algorithms by simplifying the computational world to a reduced set of parameters that define the cost of arithmetic and memory access operations and communication. These models contribute to the search for efficient algorithms for a given architecture, improving the productivity of designers, programmers, and engineers. A small amount of communication, a few operations, and a high degree of parallelism are fundamental points that directly contribute to the efficiency of a parallel algorithm.

This section summarizes the characteristics of the most widely used parallel computing models for performance estimation. It is not aimed at providing a comprehensive presentation or a thorough classification of parallel models, languages, and architectures.

3.2.1 Random access machine and parallel random access machine

The random access machine (RAM) model is proposed in [97] for sequential algorithms. It comprises a memory, control unit, processor, and program. In 1978, Fortune and Wyllie proposed the parallel random access machine (PRAM) model [98] based on the RAM model. The main idea behind PRAM is that there is a shared memory m connected to several processing units with a global clock, as shown in Fig. 3.5. In this scenario, one processor P can execute one operation (arithmetic, memory access, or logic) within one clock cycle. However, this model does not consider the communication or synchronization overheads.

PRAM sub-models like the exclusive read exclusive write (EREW), exclusive read concurrent write (ERCW), concurrent read exclusive write (CREW), and concurrent read concurrent write (CRCW) were introduced to handle read/write operations in a shared memory model [99].

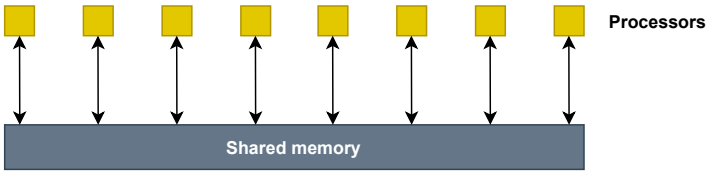


Figure 3.5: PRAM model. Different processors execute read and write operations in a shared memory. From [1].

3.2.2 Bulk Synchronous Parallel model

The bulk synchronous parallel model (BSP) [100] proposed for distributing computing is a bridging model between hardware and algorithms that offers a high degree of abstraction. The BSP program is divided into supersteps separated by a barrier synchronization. Each superstep comprises several blocks of computation and communication. Fig. 3.6 shows the workflow of the BSP model.

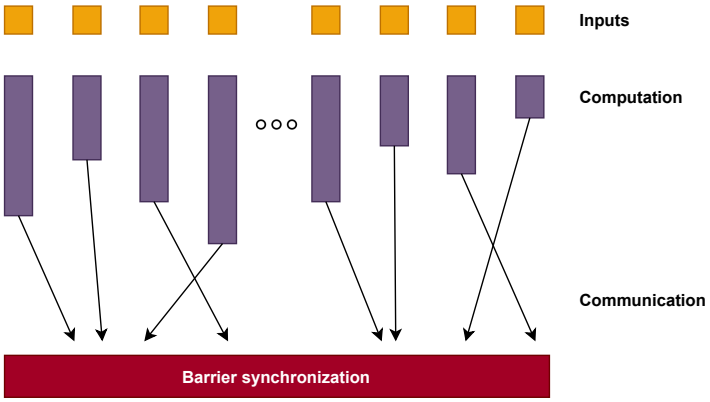


Figure 3.6: Superstep of the BSP model. From [1].

A BSP computer is represented by parameters P, s, L, and G, where:

- P: number of processors of the BSP computer.
- s: processor speed.
- L: cost, in step, to complete a barrier synchronization.
- G: cost, in words, of delivering a message.

The normalized cost G is defined by Eq.3.2

$$G = \frac{Op_{local}}{W_{sec}} \tag{3.2}$$

Where Op_{local} is the number of local operations executed per second in a processor and W_{sec} is the number of words the network communicates per second. L represents the barrier synchronization cost at the end of each superstep.

The superstep cost depends on the synchronization, computation, and communication costs of each processor of the BSP machine [101].

The multi-BSP model [102] extends the BSP to multicore architectures by considering the architecture as a tree with d leaves. Multi-BSP is a multilevel model with explicit parameters for the number of processors, memory/cache sizes, communication, and synchronization costs. The multi-BSP allows: (i) modelling a multicore computer as a tree, (ii) designing a parallel algorithm as a single program multiple data (SPMD) program with strict separation between computation and communication, and (iii) computing the cost of an algorithm on a specific computer based on computation, data movement, and latency. For a tree with i levels, the main parameters related to this model are as follows:

- P_i: number of processors at i-th level.
- g_i: communication bandwidth.
- L_i: cost, in step, to complete a barrier synchronization at level i.
- m_i: words of memory at i-th level.

BSP and multi-BSP have been widely used in multiple contexts and applications because of their flexibility in allowing portable and efficient parallel programs for a wide range of computers [103–109]. The results presented in [110] demonstrate the feasibility of the BSP-based machine learning (ML) computing model in intrusion detection. An elastic BSP for relaxing the synchronization stage in distributed deep learning is presented in [111]. The authors focus on the data parallelism approach, in which weight synchronization during training is crucial. The BSP is adapted for CUDA applications in [112]. This BSP for the CUDA model allows the prediction of execution times for a single kernel function on the GPU. This proposal focuses on several computational and communication steps but removes synchronization at the end of each step.

3.2.3 LogP model

The LogP model [113] describes a parallel machine using four main parameters: communication delay (L), communication overhead (o), the gap between each message (g, from a local point of view), and the number of processors (P). Fig. 3.7 presents a graphical representation of the different parameters. The model decomposes each communication step into three elements: L, o, and g, measured in clock cycles, but it does not include a model for application/computation. LogP is devised for distributed computation based on message passing and can simulate a BSP model.

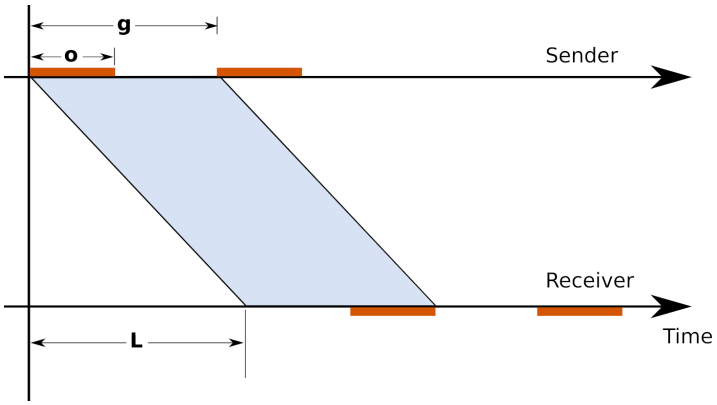


Figure 3.7: LogP model, based on [114]. From a local point of view, for one Processor (P), g represents the gap between messages, o is the communication overhead, and L is the communication delay. From [1].

Different variants of LogP, such as LogGP [115], LogGPC [116], LogPQ [117], PLogP [118], mPlogP [119], and mHLogGP [120] were introduced to improve the model.

3.2.4 Collective Computing Model

The collective computing model (CCM) [121] is based on the BSP model and is composed of processors, memory, and two types of supersteps: normal and division. The normal superstep is characterized by computation, followed by the execution of a collective communication function (f). The division superstep considers that the machine can be divided into submachines. Based on this assumption, several steps are performed: P processors are divided into r groups and the input data are distributed in tasks, each one is executed, followed by a phase of re-joinment. Finally, the distribution of the results is performed.

CCM has as parameters P: number of processors, **F**: group of collective functions f, **TF**: cost

functions for each $f \epsilon$ **F**, **P**: group of partition functions p, and **TP** cost functions for each $p \epsilon$ **P**.

3.2.5 Roofline Model

The Roofline [122] is a throughput-oriented performance model for auto-tuning the performance of multicore computers. It provides information about data movement and computation to understand the limitations of the code and combines bandwidth, locality, and different parallelization paradigms. Fig. 3.8 shows the output of the model, which includes the computational intensity, peak computation (PC), peak memory bandwidth (PMB), and architectural and algorithmic features. The main parameter of the Roofline model is the arithmetic intensity (or computational/operational intensity – CI – [GFlops per byte]), which corresponds to the x-axis and is defined as the ratio of the number of operations (floating-point) to the total data movement (bytes). The attainable performance (AP) is defined by Eq. 3.3, and corresponds to the y-axis [GFLOPS per second]. Some contributions in the literature, such as [123, 124], extend the Roofline to cache hierarchy (hierarchical Roofline) by considering L1, L2, device memory, and system memory bandwidths.

$$AP[GFLOPS/sec] = \min \begin{cases} PC, \\ PMB \times CI \end{cases} \tag{3.3}$$

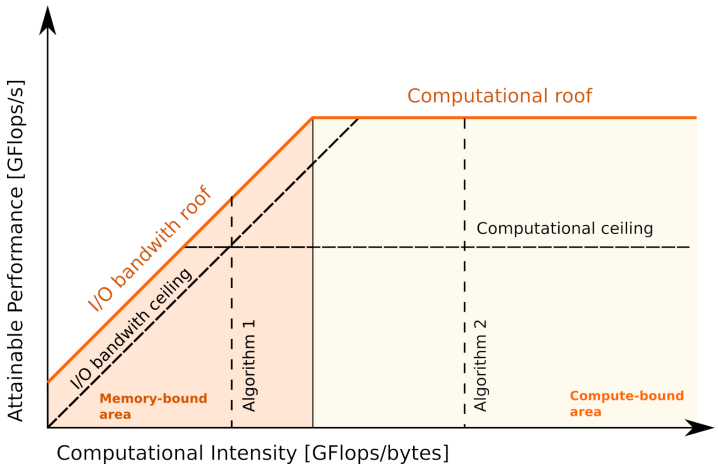


Figure 3.8: Roofline model, based on [125]. The x-axis represents the operational or computational intensity (CI) and y-axis represents the attainable performance (AP) or throughput. Computational roof and I/O bandwidth roof limit the achievable AP. On the right (yellow area), the algorithms are compute-bound, while on the left (orange area), they are memory-bound. From [1].

Roofline model for FPGA

Roofline model has been introduced in the field of SoC-FPGA to assist the designer in creating efficient hardware for HPC applications, explore the design space, and estimate the performance of ML-based models implemented on FPGA. Silva et al. [126] adapted the model for reconfigurable hardware accelerators, including scalability as a parameter to obtain the processing element (PE) replication factor based on resource utilization. The unit introduced for performance operation is byte-operations (BOPS). The parameters used by the model according to [126] are the scalability (SC), computational performance (CP), computational intensity (CI), and peak memory bandwidth (PMB). According to Silva et al., the attainable performance is defined by Eq. 3.4 when targeting FPGA/SoC devices.

$$AP[BOPS] = \min \begin{cases} CP_{PE} \times SC, \\ PMB \times CI \end{cases} \tag{3.4}$$

The *SC* of the system is defined as the ratio between the total resource available in the reconfigurable architecture and the maximum resource utilization by a given PE, thus the *SC* of one PE is constrained by the most utilized resource, as it is shown in Eq. 3.5.

$$SC = \min \left\{ \frac{BRAM_T}{BRAM_{PE}}, \frac{DSP_T}{DSP_{PE}}, \frac{LUT_T}{LUT_{PE}}, \frac{FF_T}{FF_{PE}} \right\} \tag{3.5}$$

The computational performance (CP_{PE}) for a given PE is the ratio between total number of operations and the runtime of the instance, according to the Eq. 3.6.

$$CP_{PE} = \frac{Operations}{Runtime} \tag{3.6}$$

3.3 Summary

Parallel computing models aim to bridge the gap between application and architectural domains. Table 3.1 presents a comparison of the main features of the models described in Section 3.2. The table includes the type of communication supported by the model (shared, distributed, or hierarchical), the different costs considered by the model (synchronization, asynchronous communication, computation, or memory), and the parameters used in each model.

Model	Communication			Costs				Parameters
	Shared	Distributed	Hierarchical	Synchronization	Asynchronous communication	Computation	Memory	
PRAM	x	-	-	-	-	-	-	P, m
BSP	-	x	-	x	x	x	-	P, s, L, G
LogP	-	x	-	x	x			L, o, g, P
CCM	-	x	-	x	x	x	-	P,**F**, **TF** ,**P**, **TP**
Multi-BSP	-	x	-	x	x	x	x	P_i, g_i, L_i, m_i
DRAM-only Roofline	-	-	-	-	x	x	-	CI, AP

Table 3.1: Features of the computing models PRAM, BSP, LogP, CCM, multi-BSP, and DRAM-only Roofline.

As can be observed, one of the main features of parallel computing models is their reduced number of parameters, making them easily adopted by the software designer. Nevertheless, except for the Roofline model, they have yet to be widely adopted for FPGA technology, benefiting hardware developers with a tool capable of modeling the architecture and predicting the main metrics related to FPGA-based hardware accelerators. One of the factors that may impact this is the inherent hardware reconfigurability of FPGA/SoC, challenging its modeling. Moreover, when working with HLS tools, the number of parameters that impact the hardware designs is greater than the ones considered by traditional computing models.

Besides these factors, a parallel computing model can be coupled with a performance estimator for FPGA, where latency and area are one of the main objective functions to optimize. Thus, the estimator provides the information needed to build the parallel model.

For heterogeneous architectures, the hardware-software co-design can be considered by performance estimators, taking into account the inherent features of different technologies, to ease the decision on which part of the algorithm should be implemented in software and which part in hardware. The performance of the overall system may be estimated by combining traditional parallel computing models presented in Section 3.2 (for the sequential part) and the contributions discussed in Section 4 (for the FPGA part). In addition, a single parallel model, such as Roofline, can be applied to both architectures.

Moreover, coarse-grain parallelism can be obtained by employing a model such as Roofline, identifying the computation-to-communication ratio. Thus, exposing the relationship between communication bottlenecks, computations, and the number of replicas, as was presented in Section 3.2.5 and demonstrated in contributions such as [125, 127].

Chapter 4

State of the art in performance estimators for SoC-based FPGA

This chapter presents the state of the art related to the topic addressed in this thesis allows knowing the scientific advances that occurred in recent years, supporting the research carried out. Section 4.1 presents the performance estimators for FPGA, considering general approaches in Section 4.1.1 and design space exploration in Section 4.1.2, with the corresponding discussion in Section 4.1.3. Since this thesis considers image analysis and other highly demanding applications, Section 4.2 introduces and discusses the models and frameworks developed for performance estimation in different research areas.

4.1 Performance estimation for FPGA

Performance estimation in the early stages of design is essential for improving the hardware designer's productivity when using HLS tools. To this end, studies have demonstrated an increasing trend in developing models, methodologies, and frameworks for estimating performance metrics associated with FPGA/SoC. Two categories are proposed in this chapter: general approaches and design space exploration (DSE). The former includes the proposed estimators for the area, resource

This chapter is based on the work published in [1]: **Molina, R.S.**; Gil-Costa, V.; Crespo, M. L; Ramponi, G. (2022) "High-Level Synthesis Hardware Design for FPGA-based Accelerators: Models, Methodologies, and Frameworks". In IEEE Access, vol. 10, pp. 90429-90455, 2022. IEEE.

utilization, and power consumption without considering a DSE engine. In contrast, the latter considers the exploration of the design space in addition to performance estimation.

4.1.1 General approaches

Traditional parallel computing models have been used to perform metric estimations, being the Roofline model the one with more impact in the current literature. BSP was employed in [128], while RAM was used [129]. Regarding Roofline, contributions in [125,130,131] exploited this model to compute the attainable performance of the hardware accelerators.

Kapre *et al.* [128] presented a communication discipline inspired by synchronous dataflow [132] and BSP computational models for OpenCL pipes in FPGA devices, considering that one of the strategies to exploit FPGA wiring is through pipes, by reducing the communication latency between kernels.

Hora *et al.* [129] proposed pipelining circuit RAM (PCRAM), which is a computational model that considers only synchronous circuits. Several algorithms were described, and the model was used to obtain time complexities, leaving the contrast with the experimental results for future work. In this model, the computer comprises a word-RAM of word size w with a circuit composed of an execution module, gates, and inputs/outputs.

The Roofline model is currently used to recognize the FPGA's highest performance and potential bottlenecks owing to its intuitiveness and simplicity while providing insights into arithmetic computation and attainable performance. An extended version of the Roofline multicore model for hardware accelerators was presented by Silva *et al.* [125], maintaining the core of the original proposal but adding the resource utilization and parameters obtained through HLS tools. The unit for the performance operation is byte-operations (Bops), considering that fixed-point operations are more suitable for this technology than floating-point operations. The authors also included the scalability parameter in determining the PE replication factor, considering the available resources and resource utilization per PE. Based on this initial proposal, contributions in the literature [130, 131] extended this model to FPGA devices. Calore *et al.* [130] presented an FPGA empirical Roofline (FER) to estimate the throughput and memory bandwidth of FPGAs for high-performance computing (HPC) applications based on HLS tools. Nguyen *et al.* [131] extended the empirical Roofline toolkit (ERT) to FPGAs and presented a benchmark for energy efficiency.

HLScope [133] is a performance debugging methodology that helps to identify potential bottlenecks and their causes. HLScope has two flows: in-FPGA (accurate analysis) and software simulation (rapid analysis). For each hardware described by the designer, the tool provided execution times and analyzed various stall causes: external DRAM access, synchronization, and dependency. HLScope+ [134] extended HLScope to overcome its main drawbacks. HLScope+ includes a fast and accurate HLS-based cycle estimation and an improved memory access model that considers some PE in the FPGA connected to an external memory through a DRAM controller, avoiding cache modeling.

A cost model for FPGA partial reconfiguration, proposed by Papadimitriou *et al.* [135], considered all physical elements involved in the reconfiguration process, where each phase contributed to the total reconfiguration time. The authors also explored the parameters that affect the reconfiguration performance.

FlexCL, introduced by Wang *et al.* [136], is an analytical performance model that uses the OpenCL kernel as the input and provides the performance estimated for the FPGA. The input source code was transformed into an LLVM IR trace by using Clang. Code structure and operation latency are extracted using a kernel analyzer and sent to computation, communication, and global memory models. Because of the integration of these three models, the execution time for a given kernel was estimated. FlexCL contributes to identifying performance bottlenecks in FPGA, where PEs, computation units, and kernels have their models. FlexCL considers eight global memory access patterns and can be used to explore the design space to identify solutions under given user constraints.

Pyramid, developed by Makrani *et al.* [137], is a machine learning-based framework to estimate timing and resource utilization and to overcome the differences between the post-implementation results and intellectual property (IP) cores created using HLS. It was developed by employing ensemble machine learning techniques such as linear regression, artificial neural networks, support vector machines, and random forests. As part of the framework, Minerva [138], an automated hardware optimization tool based on a heuristic model, was used to obtain a good throughput and throughput-to-area ratio for the RTL code generated by HLS. Wang *et al.* [139] presented a framework based on a performance analysis model combined with code tuning techniques for OpenCL applications only on FPGAs, assuming that designers adopt an incremental development model [140]. The model included four FPGA-centric metrics to detect possible bottlenecks related to memory, parallelism, and computation.

Power consumption is an important topic, especially with the growth of green technology, internet of things (IoT) systems, and the expansion of communication networks. Power estimation techniques are categorized based on the abstraction levels of the FPGA design process as follows: system, RTL level, gate, and layout levels. One of the requirements when designing IP cores under power, energy, or thermal constraints is their estimation during the first steps of the design process for a given application. A survey on power consumption in FPGA and ASIC devices [141] classified the techniques for its estimation into analytical, table-based, polynomial-based, and neural networks.

Neural networks have been used to estimate power consumption in contributions such as [142–144], showing the effectiveness of these techniques.

Verma *et al.* [145] presented a power estimation model that improved Deng's model [146] and was designed using nonlinear regression techniques. For this purpose, they used the power data from different types of digital circuits (described in VHDL) after the synthesis process. The data were divided into designs with and without clock gating, and based on this separation, two power models were developed.

KAPow, proposed by Davis *et al.* [147], is an online activity-based power methodology that includes a signal pruning strategy. The flow has two phases: signal selection (nets with solid relationships between activity and power) and instrumentation (implying the accumulation of events to monitor relevant signals). A linear model is used to estimate the overall system's power contribution by computing each IP core's power consumption.

FlexCL was extended in [148] by incorporating three modes of communication in the memory model: direct, burst, and stream access patterns, and an analytical power model for dynamic and static power.

HLSPredict, developed by O'Neal *et al.* [149], is a framework based on an ensemble of ten machine learning models to predict performance and power consumption without analytical models or HLS-in-the-loop. Two types of IP cores were considered: without directives (base IP core) or with directives (optimized IP core). Accelerators for training the models are based on a template with DMA for memory transactions, which implies that for every source code implemented through the HLS, the functionality of the IP core is encapsulated and integrated within the hardware template.

HL-Pow, proposed by Lin *et al.* [150], is based on machine learning techniques and overcomes the gap between the HLS synthesis phase and power consumption estimation (usually performed

after the RTL implementation flow). A DSE is introduced to obtain the latency vs power trade-off, with pruning to reduce the design space when finding Pareto-optimal designs. For the machine learning implementation, the training dataset was constructed by a feature construction (HLS report) and power collection (post-implementation report), with a total of 256 elements per feature. The experiments were performed using different machine learning models, including linear regression, support vector machines, tree-based models, and neural networks.

PowerGear, described by Lin *et al.* [151], is a graph-learning-assisted power estimator for FPGA HLS that is composed of a graph construction flow and a power-aware graph neural network model called HEC-GNN. This study considers the impact of interconnections in hardware design that affects power modelling. The authors benefit from the HLS front-end and HLS back-end to recover dataflow graphs because it is possible to obtain IR traces and finite state machines with datapath information. PowerGear can be used to guide a design space explorer with a trade-off between latency and power to obtain the Pareto frontier.

Aladdin, introduced by Shao *et al.* [152], estimates the performance, power, and area of accelerators. It generates a dependence graph from the input code and produces a fast cycle estimate before the RTL construction.

HAPE, presented by Makni *et al.* [153], is a framework for area-power estimation based on analytical models, and it aims to assist the DSE in reducing HLS runtime. HAPE focuses only on the main subtraces presented in a source code containing the directives provided by the designer. HAPE integrated LinAnalyzer for computational cost.

4.1.2 Design space exploration

Design space explorers aim to minimize HLS tools execution times, which are highly dependent on the size of the space to be analyzed. Different approaches have been proposed based on the analysis of HLS directives, where the exploration of the design space [154, 155] is important because it increases exponentially with the use of directives. The challenge is to find a set of hardware designs, also known as Pareto-optimal designs. Considering that there is a limited number of resources (LUT, BRAM, DSP, and FF) available in the reconfigurable architecture, the hardware design cannot request more resources than those available in the FPGA.

Surveys related to this topic are presented in [28, 71]. In particular, the last one proposed a

classification of HLS DSE techniques into two groups, as depicted in Fig. 4.1: synthesis-based and model-based. In this classification, the third category is composed of a combination of supervised learning and DSE synthesis-based techniques.

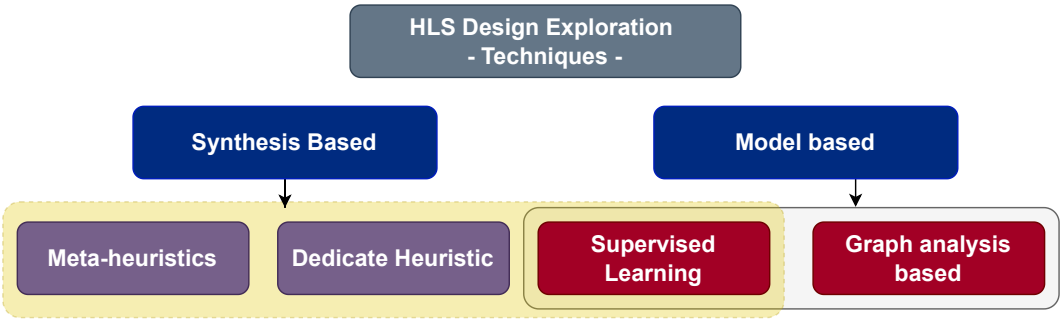


Figure 4.1: Classification of HLS DSE techniques, based on [28].

According to [28], supported by Fig. 4.2, DSE based on meta-heuristics techniques presented the highest accuracy and is easy to implement, port, and maintain. Nevertheless, they lack respect for runtime because most use HLS tools in the loop. In addition, techniques based on supervised learning exhibited equilibrated behavior along the five features presented in Fig. 4.2.

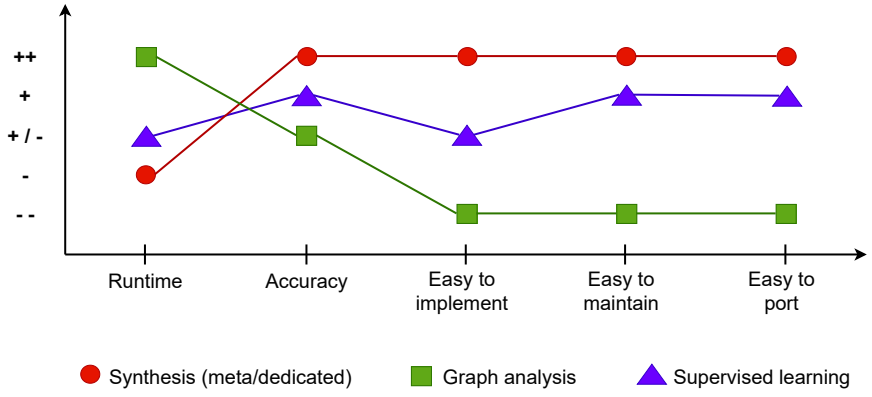


Figure 4.2: Value curve for DSE. Based on [28].

Ferretti *et al.* [156] proposed a method to infer knowledge from past design explorations. The authors introduced signature encoding for code and directives, composed of specification encoding (SE), configuration space descriptor (CSD), and similarity metric longest common subsequence (LCS). The methodology uses signature encoding to create a string with design and configuration spaces (directives and their modes) by combining CSD and SE. The LCS metric was used

to measure the similarity between the actual and previous DSE stored in a database.

COSMOS, which is an automatic and scalable methodology for DSE, was introduced by Piccolboni *et al.* [157] for complex accelerators. It generates a set of Pareto-optimal designs and reduces the number of HLS invocations. It comprises two main phases: component characterization and DSE (based on two steps: synthesis planning and mapping). The computing model used for DSE was based on timed marked graphs. COSMOS includes memory as part of the DSE process and applies synthesis constraints to reduce the variability of the HLS tools.

Lo *et al.* [158] proposed a sequential model-based optimization using a transfer-learning mechanism to select directive configurations in HLS, minimizing the number of tool evaluations/executions while obtaining solutions with LUTs-latency optimal trade-offs.

Kwon *et al.* [159] proposed a mixed-sharing multidomain model for reusing the knowledge obtained from previous HLS DSE while exploring a new target design space, demonstrating its effectiveness when approximating the quality of results (QoR) without running HLS tools.

Dai *et al.* [160] presented a fast and accurate QoR estimation based on HLS. For this purpose, they used final HLS reports from a set of synthesized applications to identify relevant features and metrics. They also constructed the dataset for training machine learning models (linear regression, artificial neural networks, and gradient tree boosting). The authors employed information from HLS reports for different directives and targeted different FPGA platforms to create the dataset. In addition, C-to-bitstream flow for different clock periods is performed to obtain features, such as post-implementation resources and the worst negative slack. Finally, the authors obtained 234 features, which were reduced to 87 after an elimination process to remove irrelevant features.

Lin-Analyzer [161] is a tool that allows accurate and fast FPGA performance estimation and DSE, considering fine-grained parallelism. With this framework, the runtime scales linearly while increasing the design space complexity; however, only a few optimizations are considered, mainly loop unrolling, loop pipelining, and array partitioning. Regarding resource utilization, the authors assumed that DSP and BRAM are bottlenecks in accelerator design. The communication cost between the FPGA and global memory was not considered. The framework is divided into three main stages: instrumentation, optimization of dynamic data dependence graph (DDDG) generation, and DDDG scheduling. In the last stage, latency was used as a performance metric under resource constraints. Lina was proposed in [162] as an extension of LinAnalyzer, including non-perfect loop nests and timing analyses.

MPSeeker was proposed by Zhong *et al.* [163] to estimate the performance and resource utilization of a given code (C/C++), considering fine- and coarse-grained parallelism, allowing fast DSE. Because MPSeeker contemplates multi-parallelism using the loop tiling technique, a gradient-boosted machine is proposed to obtain an accurate resource model for FF and LUT. In contrast, Lin-Analyzer is used for BRAM and DSP estimation. The authors also extended the features of the LinAnalyzer by including the data communication cost. The performance cost in MPSeeker is modelled as the sum of the kernel computation and data communication costs.

Choi *et al.* [86] presented a DSE and clock cycle estimator using HLS, including code transformations in the presence of variable-loop bounds. They proposed a resource prediction method based on HLS reports through shareable and non-shareable operators from a loop. Using linear interpolation, non-shareable resources are obtained, whereas the resources estimated for shareable operators are computed as the maximum of all loops. An analytical model is proposed for clock cycle prediction. In this framework, the design with the best performance is the output.

Ferretti *et al.* [164] presented a framework for HLS DSE using a cluster-based heuristic, integrally developed in MATLAB. The algorithm identifies different clusters in the DSE by reducing the number of regions to be analyzed. Intra-clustering was performed, followed by inter-cluster exploration. A lattice-traversing DSE framework [165] was proposed to explore the design space by transforming it into a lattice representation. The framework includes lattice creation and initial sampling, selection of lattice Pareto neighbors, and synthesis and lattice labelling.

COMBA [85, 166] is a framework that focuses on selecting the optimal configuration of directives in an HLS, taking into account the use and availability of hardware resources, and provides an estimation of performance and resource utilization. The authors proposed a metric-guided DSE II (MGDSE-II) algorithm to prune and explore the design space based on three metrics: the number of DSP, BRAM, and LUT. An overview of COMBA, which comprises a recursive data collector, analytical models (latency and resources), and DSE, is presented in Fig. 4.3. In COMBA, the input is C/C++ source code, which is transformed into an LLVM IR trace through Clang. The IR trace is the input for the recursive data collector, which extracts static and dynamic information that will be used for the analytical models. The MGDSE-II then evaluates the configuration and establishes the subsequent directives to be applied to the input code. This iteration is repeated until a high-performance configuration is obtained.

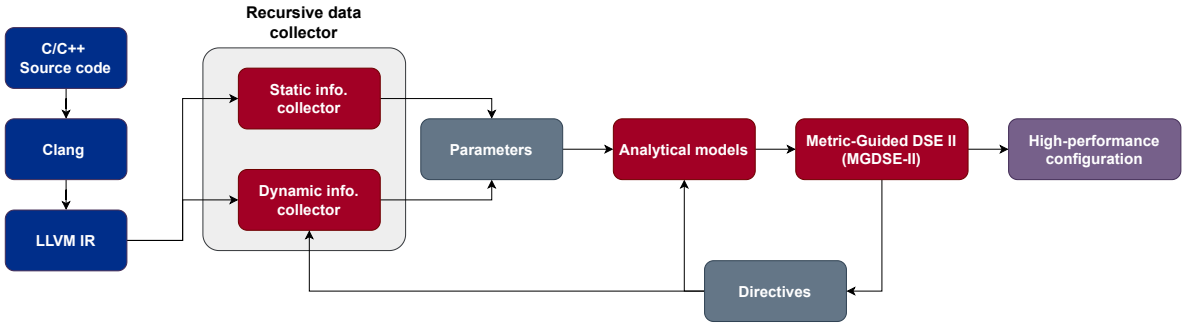


Figure 4.3: COMBA framework overview, based on Zhao *et al.* [85]. LLVM IR is extracted from the source code. This trace is the input for the recursive data collector, which will extract the parameters used by the analytical models (latency and resource). MGDSE-II evaluates the configuration and defines the next set of directives to be applied. The output of the complete flow is the high-performance configuration. From [1].

IronMan [167] is an end-to-end flexible and automated framework for DSE composed of a performance and resource predictor based on a graph-neural network (GPP), a multi-objective DSE engine based on reinforcement learning (RLMD), and a code transformer (CT). One of the main features of this framework is that it retrieves the final code with the discovered optimizations and is ready to generate the corresponding RTL through HLS.

The Roofline model was introduced within the methodologies to explore the design space, targeting HPC applications based on HLS [127, 168, 169].

Nabi *et al.* [169] proposed a TyTra flow that integrates performance and cost models based on Roofline analysis to obtain an optimized FPGA solution for scientific HPC applications. The methodology adopts the models defined in the OpenCL standard: platform and memory hierarchy, kernel execution, memory execution, and data patterns. The Roofline model is the basis for the design space explorer and is used to assist in the selection of the best instance to be downloaded into hardware. Additionally, the authors proposed an intermediate representation language (TyTra-IR). For the calculation of resource utilization to obtain scalability of the system, the authors considered a maximum utilization of the FPGA of 80%, as suggested by [170].

Siracusa *et al.* [127] proposed DSE methodology. The system input was the C/C++ source code, translated to an LLVM IR trace to obtain the baseline performance estimation and resource utilization through the synthesis process. The Roofline model chart (RooflineOrig) determines memory bottlenecks from this base implementation. Subsequently, an automated DSE estimates the re-

sources and performance and generates optimal design points. The Roofline for the best feasible design is plotted along with the RooflineOrig chart to compare the current design's performance with the solution derived by the DSE. The explorer includes resource sharing and HLS-specific IR optimizations during sample estimations. This work was extended in [168] with the hierarchical version of Roofline, estimating peak performance analytically, and integrating a guide to reaching memory transfer and data-locality optimizations.

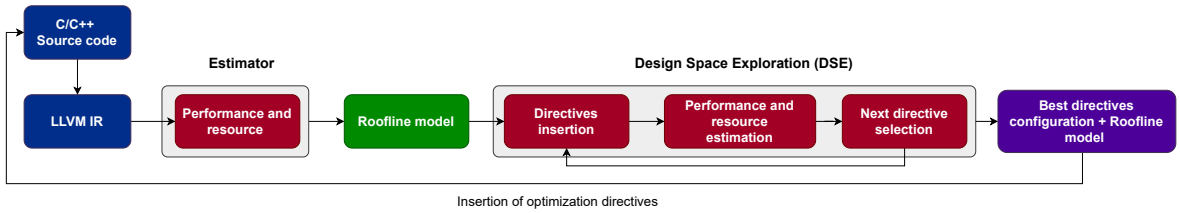


Figure 4.4: A DSE methodology presented in [127, 168]. The input source code is translated to LLVM IR trace, obtaining the baseline for performance estimation and resource utilization. Subsequently, the Roofline model chart estimates memory bottlenecks. An automated DSE phase allows resource and performance estimations, and the best feasible design is plotted along with the original Roofline chart. From [1].

One of the fundamental points when designing DSE engines for SoC-based FPGA is the time required to sample the search space because it is usually linked to the execution of HLS tools in the loop. Contributions in the literature aimed to reduce the search space [156–158, 160, 164, 171–173].

Based on the premise that search space exploration is a time-consuming task and that its complexity is associated with different combinations of directives, user constraints, and technology features, DSE has been treated as a black-block optimization problem [71].

In the context of evolutionary algorithms (EAs), several contributions in the literature have aimed to integrate the HLS tool into the loop by combining the exploration stage with EAs. Schafer *et al.* [174] proposed a combination of NSGA-II and a machine learning-based model to predict the performance. Nevertheless, the HLS tool was incorporated into the DSE flow. Moreover, NSGA-II was employed in [175, 176] using HLS in the loop. Nevertheless, EAs based on NSGA-II for DSE has yet to be used again in recent years, exploiting this solver in combination with an accurate performance model, avoiding the full use of HLS in the loop, except for the final evaluation at the end of the exploration process.

Based on Bayesian optimization, Sherlock [173], introduced by Gautier *et al.*, is a DSE frame-

work based on multi-objective optimizations devoted to finding Pareto-optimal solutions (or Pareto fronts) and handling multiple conflicting optimization objectives. This framework uses active learning to exploit a surrogate design space model to determine Pareto-optimal designs as quickly as possible. Chimera [177] is a multi-objective DSE based on active learning and evolutionary algorithms. One objective was to reduce the number of HLS invocations during the exploration process.

Mehrabi *et al.* proposed Prospector framework [178] that uses Bayesian techniques to obtain the best configurations with fewer resources and reduced latency near Pareto-efficient designs. The HLS tool is considered a black box (or function) that must be modeled and optimized. Prospector is shown in Fig. 4.5, where the inputs are the source code, clock frequency, and directives, and the outputs are the synthesized designs. The Bayesian optimization unit (BOU) was used to explore the design space and control the selection of directives. The HLS tool was used to generate the RTL from high-level source code. At the end of the process, the framework can obtain different designs with a latency-area trade-off belonging to the Pareto frontier.

Others contributions devoted to DSE are introduced in [171–173, 179–181].

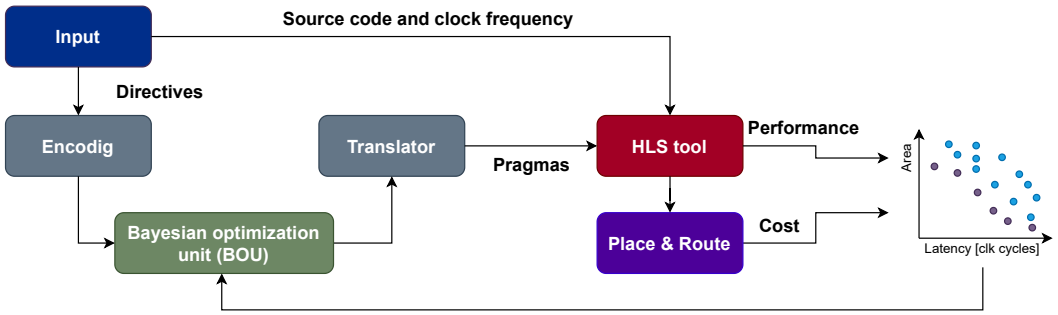


Figure 4.5: Prospector framework, based on [178]. The inputs are the source code, clock frequency, and directives; and the outputs are the synthesized designs with a trade-off between latency and area. The directives are encoded and sent to the BOU. Source code and clock frequency are the inputs for HLS Tools. Performance and cost values are obtained from HLS tool and Place & Route process. From [1].

4.1.3 Discussion

Table 4.1 shows that the described contributions (from 2016 to 2022) are fairly distributed between models (34%) and frameworks (42%), whereas 24% propose methodologies. In line with the growing tendency in developing design space explorers, 56% of the contributions include DSE.

						Metrics				Input					Optimized designs		Technique			
Paper	**Year**	**Model**	**Methodology**	**Framework**	**DSE**	**A**	**L**	**P**	**Other**	**S-C**	**C/C++**	**I-C**	**HDL**	**OpenCL**	**Pareto**	**HP Config**	**Statistical**	**Analytical**	**ML**	**Others**
[125]	2013	x (10+)							C		x							x		
[152]	2014			Aladdin			x	x			x						x			
[161]	2016			Lin-Analyzer	x	x	x				x							x		
[171]	2016			x	x	x	x				x					x		x	x	
[139]	2016			x					C					x				x		
[180]	2016		x		x	x	x							x	x				x	
[158]	2016	x (-)			x	x	x				x					x	x			
[142]	2016	x (-)						x					x			x	x			
[85]	2017			COMBA	x	x	x				x					x		x		
[169]	2017		TyTra		x	x	x		C					x						
[163]	2017			MPSeeker	x	x	x		C		x							x	x	
[133]	2017		HLScope			x	x		C		x							x		
[134]	2017		HLScope+				x		C		x							x		
[136]	2017	FlexCL (10+)				x	x		C	x				x				x		
[157]	2017		COSMOS		x	x	x			x					x					PN
[86]	2018			x	x	x	x				x					x		x		
[182]	2018	x (-)						x				x						x		
[153]	2018			HAPE		x		x			x				x			x		
[148]	2018	FlexCL (21)				x	x	x	C					x				x		
[147]	2018		KAPow					x						x					x	
[165]	2018			x	x	x	x				x				x				x	
[164]	2018			x	x	x	x				x				x				x	
[129]	2018	x (-)					x						x					x		
[160]	2018	x (10+)					x		QoR	x	x								x	
[149]	2018			HLSPredict			x			x	x								x	
[162]	2019			Lina	x	x	x				x							x		
[137]	2019			Pyramid		x	x		T	x									x	
[183]	2019	x (-)			x	x	x			x	x				x					
[179]	2019			XPPE	x		x		S		x								x	
[184]	2019		x		x	x		x			x				x				x	
[145]	2019	x (10+)						x	E				x					x		
[144]	2019	x (11)						x	E				x				x		x	
[181]	2019		x												x					
[159]	2020	x (-)			x	x	x		QoR					x	x				x	
[185]	2020	x (10+)			x				C		x								x	
[156]	2020		x		x	x	x				x				x				x	
[143]	2020	x (10+)			x			x			x								x	
[150]	2020			HL-Pow				x			x				x				x	
[127]	2020		x		x	x	x		T, C		x				x			x		
[131]	2021	x(-)			x				T, C		x				x			x		
[168]	2021		x		x	x	x		T, C		x				x			x		
[130]	2021	x(-)				x	x		T, C		x				x				x	
[178]	2021			Prospector	x	x	x				x				x		x			
[167]	2021			IronMan	x	x	x				x					x			x	
[172]	2021			AutoDSE	x	x	x				x					x			x	MH
[151]	2022			PowerGear	x			x			x				x				x	
[177]	2021			Chimera	x	x	x				x				x				x	MH
[173]	2022			Sherlock	x	x	x				x				x				x	
Percentage		34%	24%	42%	56%	58%	66%	26.5%	36%	13%	66%	2%	8%	14%	34%	14%	10%	40%	44%	6%

Table 4.1: Contributions presented in the literature for performance estimation. The acronyms used in the table are: A: area, L: latency, P: power consumption, QoR: quality of result, C: communication, T: throughput, E: energy, S: speed-up, RT: reconfiguration time, S-C: SystemC, I-C: Impulse C, HDL: hardware description language, MH: meta-heuristics, Em: empirical, and PN: Petri Nets.

It is noticed that most DSE solutions use high-level abstraction languages as input, showing a tendency to increase productivity in the design phase. Likewise, many studies are focused on obtaining Pareto-optimal designs.

Regarding metrics, latency and area are the most frequently estimated metrics, followed by power: 66%, 58%, and 26.5%, respectively. We also present this result in Fig. 4.6 (1). The area and latency metrics are widely estimated because reconfigurable platforms are resource constrained and are used for algorithm acceleration.

Concerning power consumption, some described contributions highlight the benefits of estimating this metric for a given application at an early stage of its design. Some of the most recent studies benefit from HLS tools to estimate this metric before the implementation stage of the overall system into the hardware platform. This approach is becoming commonplace in the literature when considering FPGA/SoC as a development architecture.

Table 4.1 also shows that the C/C++ source code is preferably used as input (66%), and the Pareto frontier is the most applied solution to obtain optimal designs (34%) in terms of trade-off between area and latency, area and power, latency and power, among other metrics. Whereas machine learning and analytic methods are almost equally used to obtain accurate, fast, and robust models (44% and 40%, respectively), as shown in Fig. 4.6 (2). However, in the last years, machine learning has been the most widely used technique.

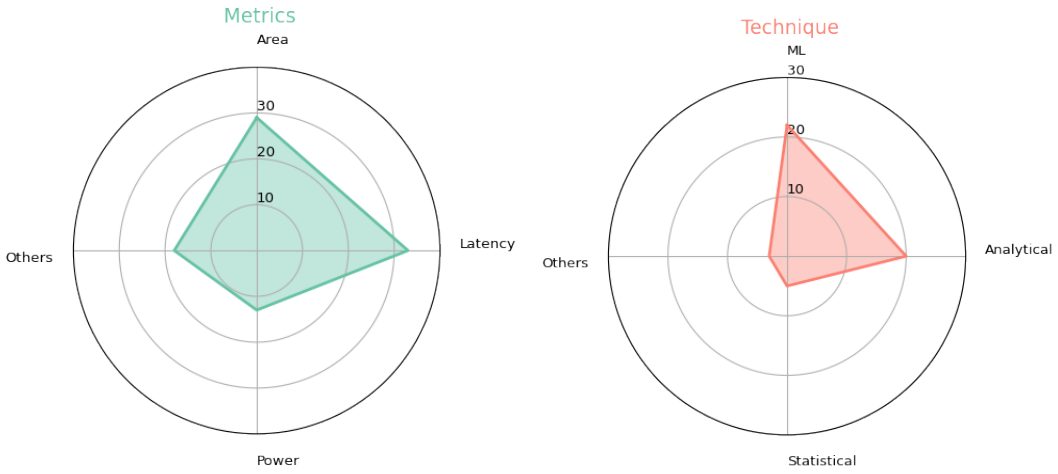


Figure 4.6: Radar plots. (1) Metrics. (2) Techniques.

The contributions devoted to models, methodologies, and frameworks for metric estimation,

FPGA-based DSE, and power consumption described in this section are illustrated in Fig. 4.7. It can be observed that, in recent years, there has been an increasing number of frameworks including DSE, whereas the power consumption is mainly estimated by models, with a preponderance of analytical techniques.

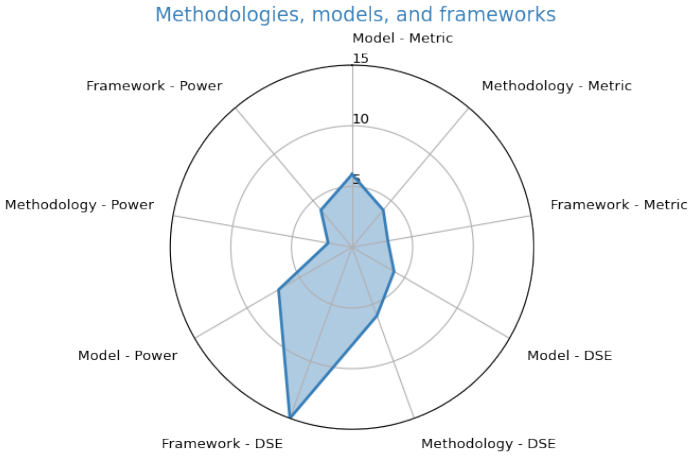


Figure 4.7: Radar plot for models, methodologies, and frameworks for metric estimation, FPGA-based DSE, and power consumption.

From the exposed in section 4.1.2, only a few contributions include more than two aspects when developing DSE. A design space explorer can benefit from reducing the design space by focusing on obtaining design points near the Pareto frontier, a parallel computing model to guide performance estimation, a reasonable estimation of QoR, and resource utilization. Transfer learning, a technique linked mainly with ML approaches, could help to obtain underlying patterns when developing hardware through HLS tools.

Some contributions only estimate some of the FPGA resources, as follows. LUT-latency trade-off is estimated by [158], and BRAM and LUT are computed by [171]. COMBA [85, 166] estimates DSP, BRAM, and LUT. Lin-Analyzer [161] computes BRAM and DSP, whereas MPSeeker [163] estimates FF and LUT, combining Lin-Analyzer for DSP and BRAM utilization. Nevertheless, overestimating resource utilization can lead to pruning valid design points in the exploration phase. LUT, FF, DSP, and BRAM post-implementation estimation is performed by [160]. A challenge with HLS tools is efficiently predicting resource sharing for unrolling factors and array partitions when using HLS pragmas. [86, 127].

4.2 Estimators for highly demanding applications

Contributions in the literature propose performance estimators for specific hardware acceleration applications, which are used in diverse research areas. Some of these are based on general models, such as Roofline.

4.2.1 Models

The Roofline model was adapted to FPGA to explore the design space, estimate the performance, and evaluate the throughput because of its dependency on communication and computation. The introduction of this model helped to assist the designer when targeting hardware acceleration of HPC applications.

Roofline was applied by Du *et al.* [186] in the acceleration of the stencil computation kernels by Karp *et al.* [187] for the hardware implementation of a spectral element method and by Nagasu *et al.* [188] in the context of an FPGA-based tsunami simulation.

In computational fluid dynamics (CFD), Du *et al.* [189] presented an FPGA-based CFD simulation architecture using a performance model to guide the DSE while achieving the maximum performance of the lattice Boltzmann method and searching for an optimal combination of the parameters of the unroll directive.

Reggiani *et al.* [190] presented the acceleration of iterative stencil computation using Verilog to describe hardware. An analytical model that considers memory transfer and computation was proposed to estimate the performance of the accelerator and accelerate the exploration of the design space.

Through efficiency degradation, it is possible to obtain hardware designs with higher performance, lower power consumption, and lower resource utilization at the cost of the QoR. Manuel *et al.* [191] proposed a DSE in the context of model-based approximate computing for image processing, using a multi-objective genetic algorithm to find a wide range of Pareto-optimal solutions, from which the desired compensation between quality and resources can be chosen.

In recent years, ML techniques have been applied in multiple fields such as fluid dynamics, high-energy physics, information retrieval, image processing, video processing, security, and biology [19–21]. Because of this trend, models for FPGA-based architectures are being developed to accelerate ML applications with the efficient exploitation of hardware resources, with the aim of

improving productivity in the design phase [22–24].

Resource and performance models were proposed by Reggiani *et al.* [192] for convolutional neural network (CNN) accelerators to drive an automatic Pareto-optimal DSE and explore network performance on different hardware platforms. These models were applied to convolutional cores, which are critical components of the design and directly affect the overall latency and DSP utilization. The final relation to obtain the Pareto-optimal solutions is the number of DSP vs the initiation interval (input rate of the pipeline in clock cycles).

Gysel *et al.* [193] presented an analytical model for deep CNN design, which is useful for obtaining the computational cost and inferring the required memory bandwidth for hardware design.

CaFPGA, developed by Xu *et al.* [194], is an FPGA-based DSE for CNN that focuses on the convolutional and fully connected layers. To improve productivity in the design phase, the authors proposed an automatic generation model, including incremental searching and flexible layer-folding algorithms, considering that on-chip memory is a limited resource in an FPGA. The analysis of the design space was performed using time, resources, memory, and performance models.

Shan *et al.* introduced [195] a CNN multi-kernel application and its implementation on AWS-F1, where an analytical model iwas used to compute data transfers (CPU to DDR, DDR to FPGA, FPGA to DDR, and DDR to CPU) and kernel computation times.

The Roofline model has been employed as a performance predictor for FPGA-based CNN accelerators [196–199]. Ayat *et al.* [196] presented the optimization of an FPGA-based CNN accelerator for energy efficiency. Xie *et al.* [197] used this model to quantitatively analyze the design phase of a CNN accelerator, depending on the available computing and memory resources. Park *et al.* [198] proposed a model based on Roofline to effectively compute convolutional layers using metrics such as throughput, on-chip memory, off-chip memory bandwidth, and computation-to-communication ratio.

Ma *et al.* [199] introduced a coarse-grained analytical performance model for CNN accelerators. For this purpose, the modelling of DRAM access, latency, and the on-chip buffer is analyzed to obtain the final model. Regarding DSE, convolution throughput is the main focus, considering factors such as the operating frequency, external memory bandwidth, and loop unrolling variables, using Roofline to analyze the throughput of the CNN accelerator. Resource costs were obtained by considering knobs loop unrolling and tiling.

Table 4.2 summarizes the models used in the contributions described in this section. The first

two columns represent the reference and the year of publication. The third column shows the research area in which the model is applied. The fourth and fifth columns are the aim and type of model used, respectively, and the last column is the target platform.

We can observe that most contributions focus on CNN accelerators and that the models are devoted to carrying out DSE and performance estimation and are mainly based on Roofline. The use of this model is based on the premise that communication and computation are two basic constraints for improving the throughput of an accelerator, especially when developing hardware for highly demanding applications.

Paper	Year	Field	Aim of the model	Type of model	Platform
[193]	2016	ML accelerators (CNN)	DSE	Analytical + Roofline	Xilinx Virtex-7
[188]	2017	Tsunami simulations	Scalability and performance	Roofline	Intel Arria 10
[194]	2018	ML accelerators (CNN)	DSE	Analytical	Xilinx Virtex-7
[197]	2018	ML accelerators (CNN)	Performance	Analytical	Xilinx XCVU9P
[190]	2018	Various	Performance	Analytical	Xilinx VC707
[196]	2018	ML accelerators (CNN)	Performance	Roofline	Xilinx Zynq
[192]	2019	ML accelerators (CNN)	Network Perf.	Analytical	Xilinx Virtex-7
[198]	2020	ML accelerators (CNN)	DSE	Roofline	Xilinx Virtex-7
[199]	2020	ML accelerators (CNN)	Performance + DSE	Analytical + Roofline	Intel Arria 10 and Stratix 10
[191]	2020	Image processing	DSE	Analytical	Intel Arria 10
[189]	2020	Fluid dynamics	Performance	Deterministic	Xilinx Alveo
[186]	2020	Fluid dynamics	Performance	Roofline	Xilinx Alveo
[195]	2020	ML accelerators (CNN)	Data transfer and computation	Analytical	Amazon EC2 F1
[187]	2021	Fluid dynamics	Performance	Roofline	Intel Stratix 10 GX2800
[200]	2021	Image processing	DSE		

Table 4.2: Models used for FPGA/SoC on different research areas.

4.2.2 Frameworks

Frameworks (or toolflows) have been proposed to map ML inference and training into SoC-based, integrating models to estimate hardware resource utilization, latency, and throughput mainly. An exhaustive survey is presented in [22].

Concerning training acceleration, Geng *et al.* [201] developed FPDeep, a toolflow for scalable CNN training acceleration on deeply pipelined FPGA clusters, proposing a model for operator graph partitioning and hardware resource allocation (with a distinction between small and large

FPGA clusters). Roofline was used to evaluate throughput because of its dependency on communication and computation.

F-CNN, introduced by Zhao *et al.* [202], is an automatic framework for CNN training based on the reconfiguration of a streaming data path at runtime. The proposed resource and bandwidth estimation models guide space exploration under design constraints to obtain an optimal performance.

HP-GNN, proposed by Lin *et al.* [203], is a framework for training graph neural networks (GNN) on a CPU-FPGA platform. It incorporates an engine dedicated to exploring the design space through an exhaustive search using performance and resource utilization models. HP-GNN also incorporates hardware templates to implement different GNN architectures.

Regarding inference acceleration, Ghaffari *et al.* [204] presented CNN2Gate, a framework based on OpenCL, to map a CNN onto an FPGA with fixed-point arithmetic, including a hardware-aware DSE based on resource utilization. It was implemented using manual directive tuning, reinforcement learning, and hill-climbing methods.

Venieris *et al.* [205] proposed the fpgaConvNet toolflow to map a CNN onto an FPGA, thereby optimizing the neural network workload. It includes a DSE using a multi-objective algorithm (simulated annealing), where the explorer optimizes the design according to the latency, throughput, or maximum throughput with a latency constraint. Performance estimation and resource utilization models were proposed for DSE.

Cloud-DNN [206], introduced by Chen *et al.*, is a framework for mapping DNN to cloud-FPGA and generating the corresponding HLS project to obtain the final IP core. The proposed accelerator model is based on the hardware resource cost (considering DSP and BRAM) and a performance model for each layer (convolutional, max pooling, and fully connected). A greedy algorithm was employed to search for the best accelerator configuration under constraints such as the DSP, BRAM, bandwidth, and DNN layers.

FRED [207], developed by Biondi *et al.*, is a framework for real-time applications that benefits from a dynamic partial reconfiguration (DPR). It includes a hardware task model for the tasks carried out by the FPGA with partial reconfiguration enabled, a software model for the tasks executed on the processor, and a scheduling infrastructure.

Mu *et al.* presented [208] a collaborative framework to obtain OpenCL-based hardware designs for CNN implementations. A DSE based on LoopTrees is generated and pruned to reduce the de-

sign space. Fine-grained and coarse-grained analytical models are introduced to generate the final optimized solution. The former estimates latency and resource utilization, whereas the latter applies further optimization to the best candidate designs obtained after applying the fine-grained model.

The heterogeneous image processing acceleration (Hippac), proposed by Reiche *et al.* [209], is a framework that allows the generation of image processing accelerators. Several steps were performed by analyzing the IR trace: data dependency analysis, dependency graph restructuring, and transformations (streaming objects, memory allocation, and replication of the innermost kernel to improve throughput).

A framework called Spark-to-FPGA-Accelerator (S2FA), introduced by Yu *et al.* [210], transforms Scala computational kernels based on Apache Spark applications into optimized accelerator designs. For this, a learning-based DSE was employed to obtain high-performance RTL designs using an ensemble of reinforcement learning algorithms: uniform greedy mutation, differential evolution genetic algorithm, particle swarm optimization, and simulated annealing. The HLS tool was executed in the loop to verify the optimization.

AutoDNNchip [211] was proposed by Xu *et al.* to facilitate fast chip designs based on DNN, targeting FPGA, and ASIC platforms. The main factors involved in the DNN acceleration process are the bit precision, clock frequency, memory technology, PE architecture, width for data transfer, memory allocation, and DNN mapping. AutoDNNchip is composed of a chip predictor and a chip builder. The former predicts metrics such as area, latency, energy, and throughput, whereas the latter performs the DSE optimizing the chip design using the results obtained by the predictor. A chip predictor is formed by two modes: (i) coarse-grained and (ii) fine-grained. In (i), analytical models are used to obtain the energy, critical path, and area for a DNN model, whereas in (ii), an algorithm is implemented to obtain the final latency through runtime simulations, considering the results of the coarse-grained mode. A chip builder is composed of a DSE based on two phases: early stage architecture and IP configuration exploration, and inter-IP pipeline exploration and IP optimization. Finally, the RTL was generated and executed to validate the results.

Table 4.3 summarizes the frameworks used in the contributions described in this section. The first two columns are the reference and the year of publication. The third column is the research area in which the model is applied. The fourth is the framework's name, and the last is the target platform.

Paper	Year	Field	Framework	Platform
[207]	2016	Real-time applications with PDR	FRED	Xilinx Zynq 7010
[209]	2017	Image processing	Hipacc	Xilinx Kintex 7 and Intel Stratix V
[202]	2017	ML accelerators (training)	F-CNN	Intel Stratix V
[205]	2017	ML accelerators (inference)	fpgaConvNet	Xilinx Zynq 7020/45
[210]	2018	Big data analytics	S2FA	Amazon EC2 F1
[208]	2018	ML accelerators	Collaborative framework	Xilinx Virtex 7
[201]	2019	ML accelerators (training)	FPDeep	Xilinx VC709 (x8)
[206]	2019	ML accelerators (inference)	Cloud-DNN	Amazon EC2 F1 and VU9P
[204]	2020	ML accelerators (inference)	CNN2Gate	Intel Cyclone V and Arria 10
[211]	2020	DNN chip design (inference)	AutoDNNchip	Xilinx Ultra96 and ASIC
[203]	2022	ML accelerators (training)	HP-GNN	CPU + Xilinx Alveo U250

Table 4.3: Utilization of frameworks FPGA/SoC on different research areas. PDR: Partial dynamic reconfiguration.

As it is observed, most frameworks are devoted to mapping ML-based inference into FPGA/-SoC architectures. The components of these frameworks are usually expressed as pre-defined optimized templates, mainly implemented in C++ and OpenCL, where parallelism can be controlled by changing the parameters associated with the different directives.

4.3 Summary of the chapter

This chapter presented and discussed the most relevant contributions in the literature related to the development of this thesis.

In particular, the previous studies have yet to comprehensively address using performance estimators based on analytical models integrated with a DSE engine based on mathematical programming and guided by HLS rules. Several attempts have been made; however, they have included the execution of HLS tools in the loop. In addition, most developments have been tested with benchmarks, which need more diversity of real-world applications.

Contributions in the literature showed the effective use of NSGA-II solver to explore the design space using heuristic approaches. Although these techniques have been used in previous contri-

butions, EAs based on NSGA-II for DSE have yet to be used again in recent years, exploiting this solver with an accurate performance model and avoiding using HLS in the loop.

Further, as observed from the literature review, DSE based on meta-heuristics methods presented the highest accuracy and are easy to implement, port, and maintain, lacking for runtime because most use HLS tools in the loop. Moreover, Bayesian optimization is one of the most used black-box optimization techniques. As heuristic approaches, the bottleneck of this type of implementation relies on the execution of HLS in the loop.

Nevertheless, several challenges still need to be addressed to adopt performance estimators in the design flow for developing efficient hardware using HLS tools, as introduced in Section 1.6.

The following chapter presents the performance estimator proposed in this thesis.

Chapter 5

MARTE: a comprehensive hardware acceleration performance estimator

This chapter presents MARTE, a comprehensive performance estimator for hardware acceleration. Section 5.1 explains the general flow of MARTE. Then, the different stages of MARTE are presented: the initialization stage in Section 5.2, the cost model for latency and resource estimation in Section 5.3, MARTE DSE engine implemented through mathematical programming in Section 5.4, and the outputs of MARTE in Section 5.5.

5.1 MARTE general flow

As presented in Chapter 4, there is a need to develop a performance estimator to be used in the early stages of hardware design, bridging the gap between the application and SoC-based FPGA architecture. To this end, Fig. 5.1 presents the schematic of MARTE, the comprehensive performance estimator for hardware acceleration proposed in this thesis. It comprises four steps: initialization, cost model, DSE engine, and outputs.

In the **initialization** phase, the source code associated with the application to be accelerated is provided by the developer in C/C++ and transformed manually into a tree data structure, thereby identifying the main parts that affect the overall system's performance. Each node of the tree contains information related to a given statement.

Then, the **cost model** is applied to the tree to achieve the performance estimation. With MARTE,

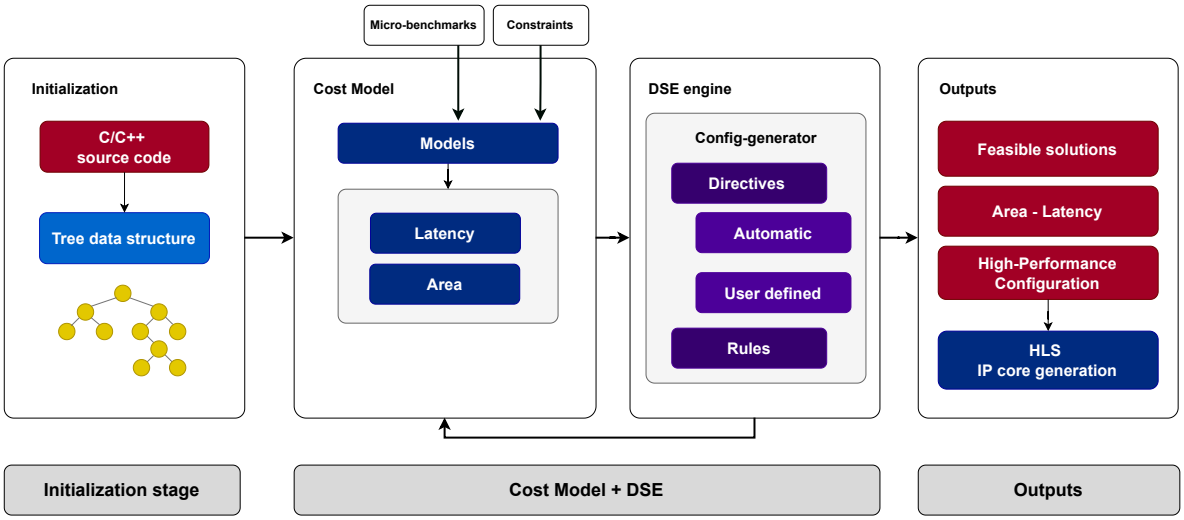


Figure 5.1: General flow of MARTE, a comprehensive hardware acceleration performance estimator.

the main objective functions estimated are the latency (L) and area, which includes hardware resources: reconfigurable hardware (LUTs and FFs) and static hardware (DSPs and BRAMs).

The next step corresponds to the **DSE engine**, which is responsible for exploring the design space for the different combinations of directives. This is an iterative process until a good trade-off between objective functions is achieved. Finally, the **outputs** of the system are the different feasible solutions, latency, resource utilization, and high-performance configurations.

MARTE reduces the time required to evaluate each solution compared with the HLS synthesis process, thus avoiding the c-to-synthesis flow to obtain area and latency estimations. Furthermore, it minimizes the time required to find a solution with a good trade-off between latency and area, improving the hardware design's performance. MARTE also helps improve productivity by providing developers with a tool that can be included in the hardware design flow while maintaining the constraints required to accelerate applications. In addition, MARTE can be used solely as an estimator or also exploiting its internal DSE engine, providing versatility to the hardware developer.

MARTE supports 32-bit floating point and 4, 8, 16, 32 bit fixed point data types. Regarding directives, pipeline, unrolling, array partitioning, and loop flattening are considered.

5.2 Initialization stage

During the initialization stage, the source code is provided by the developer in C/C++ and transformed manually into a tree data structure, as shown in Fig. 5.2. Thus, obtaining operator types and tracking activities (operations, optimizations, and dependencies) is feasible.

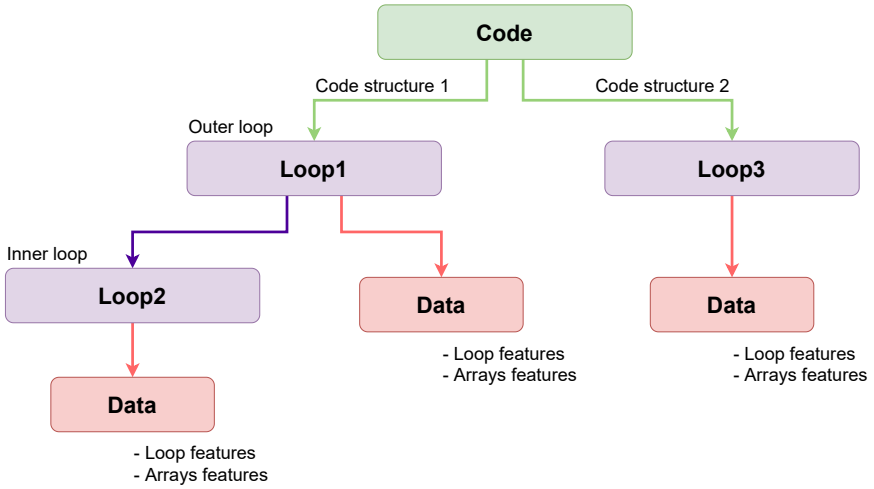


Figure 5.2: Input of MARTE: source code as tree data structure.

The tree data structure comprises a parent node that represents the source code of an application. Each tree node corresponds to a type of statement and contains associated properties. If a node represents a loop, features such as loop bounds, initiation interval, number of operations, and type of operations (addition, subtraction, multiplication, division, load, and storage) are considered, thereby maintaining the hierarchy in the presence of nested loops.

5.3 Cost Model

The cost model aims to predict the objective functions of latency (clock cycles) and area (represented by LUT, BRAM, DSP, and FF). It comprises a model for each objective function and is based on analytical and statistical techniques, considering directives for loop handling and array partitioning, which are typically employed to improve latency, performance, throughput, and area. Some parts of the cost models are inspired by COMBA [85].

The inputs of this stage are the tree data structure, micro-benchmark database, and constraints (e.g., FPGA specifications related to the family part and user specifications).

The micro-benchmark database is generated offline and composed of comma-separated values (csv) files which store the cost of basic operations (addition, subtraction, division, and multiplication), load and store operations, operation combinations, directive combinations (pipeline, unroll), and arithmetic precision (32 bit floating point, and 4, 8, 16, and 32 bit fixed points). To this end, different basic code structures were generated and synthesized using the HLS tool. The costs are defined through resources (FF, LUT, BRAM, DSP) and latency (clock cycles) for each operation and combination. E.g., in the case of 32-bit floating point operations, the resource utilization and latency obtained through micro-benchmarks and associated with multiplication, addition, and division operations are presented in Table 5.1.

Operation	**BRAM**	**DSP**	**LUT**	**FF**	**Clock cycles**
Multiplication	-	3	135	128	2
Addition	-	2	214	227	4
Division	-	-	8	49	8
Subtraction	-	2	214	227	4

Table 5.1: Resource and latency estimation for the basic operations obtained through HLS tool.

Therefore, the micro-benchmark database comprises:

- A file containing the cost of the arithmetic and memory access operations (load and store), considering 32-bit floating-point and 4, 8, 16, 32 bit fixed point data types.
- Files with the cost regarding the combination of operations inside a loop for no directives, unroll, and pipeline considering 32-bit floating-point and 4, 8, 16, 32 bit fixed point data types.

As an example, for the case of operation combinations and data types, Table 5.2 presents two rows of the data structure composition for Pipeline directive, considering the number of DSP, FF, and LUT due to expressions (expDSP, expFF, expLUT), LUT consumed by multiplexers (muxLUT), LUT and FF used as registers (regLUT, regFF), total number of operations (Nop), trip count (TC),

number of each operation inside the loop (Add, Mult, Sub, Div), and the total number of expressions (NopExp).

expDSP	expFF	expLUT	muxLUT	regLUT	regFF	Nop	TC	Add	Mult	Sub	Div	NopExp
0	0	61	36	0	10	1	5	1	0	0	0	5
0	0	54	36	0	12	2	10	1	1	0	0	5

Table 5.2: Pipeline directive: resource estimation for 32-bit fixed point, obtained through HLS tool. From left to right: number of DSP, FF and LUT due to expressions (expDSP, expFF, expLUT), LUT consumed by multiplexers (muxLUT), LUT and FF used as registers (regLUT, regFF), total number of operations (Nop), trip count (TC), number of each operation inside the loop (Add, Mult, Sub, Div), total number of expressions (NopExp).

In the following sections, II represents the initiation interval (or input rate of the pipeline), IL is the iteration latency and depends on load and store operations and on computing operations inside the loop. TC defines the trip count of the loop, which is the difference between the upper and lower bounds, UF defines the unrolling factor, and AF is the array partition factor. The latency for a loop structure is denoted as L_{clk}. In addition, for a better understanding of the parameters, Fig. 5.3 shows TC, II, IL, and loop latency (L_{clk}) in a loop without directives.

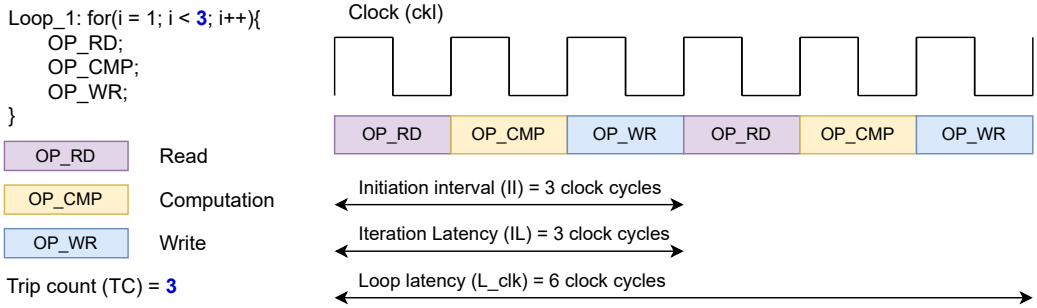


Figure 5.3: Terminology for a loop (without directives).

5.3.1 Latency model

The latency model is based on an analytical approach and aims to predict the latency in clock cycles for a given hardware design. Fig. 5.4 presents the leading considerations to perform the overall latency estimation at a macro level. In this research, estimations are considered at the macro level because the objective is to obtain an approximation of the performance in the early stages of de-

sign, avoiding micro-level approximations.

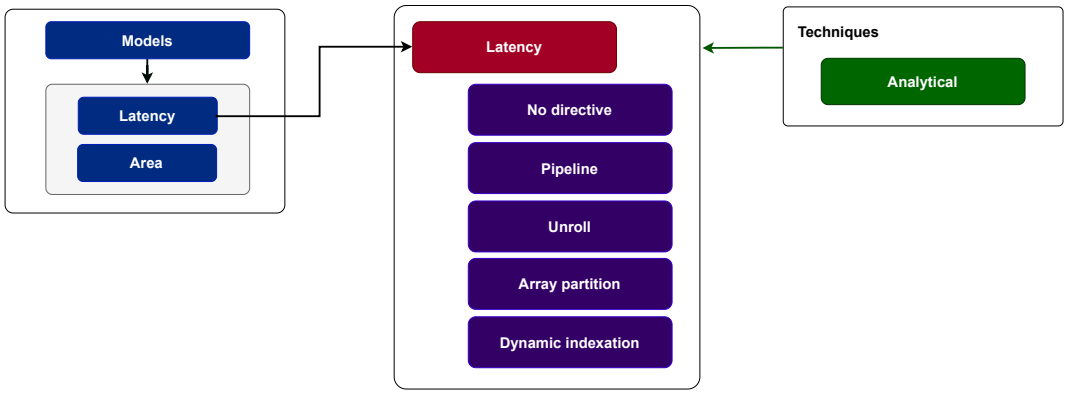


Figure 5.4: Model for latency estimation.

For latency estimation, several modes are considered based on the directives applied and operations, described as follows:

- **No directives:** the latency is computed in the absence of directives.
- **Loop pipelining:** the latency is computed when the pipeline directive is applied. The value of II can be specified explicitly by the designer.
- **Loop unrolling:** the latency is obtained when the unroll directive is applied. The value of UF can vary according to the designer's criteria.
- **Array partition:** the latency is estimated, taking into account the impact of the partition array. AF can be provided by the developer.
- **Dynamic indexation:** when an array is accessed in a non-contiguous region through a non-constant index.
- **Load and storage operations:** impact of these operations in the latency.

Therefore, **the final latency L for a given application is the sum of all the contributions of considered modes.**

No directives

The simplest case is that of the absence of directives. Latency L_{clk} for a single loop is defined by Eq. 5.1, directly associated with the clock cycles consumed for each operation inside the loop ($IL \times TC$), considering the initiation interval of the loop (II).

$$L_{clk} = II + IL \times TC \tag{5.1}$$

The latency associated with the computing operations inside the loop L_{op} (therefore, IL_{op}) is defined by Eq. 5.2, where nOp_i represents the number of specific operations (addition, multiplication, division, and subtraction), LOp_i the latency corresponding to each operation nOp_i inside the loop, the subindex i specifies the operations (e.g., if 4 operations are considered, then 0: addition, 1: multiplication, 2: subtraction, 3: division)

$$L_{op} = IL_{op} = \sum_{i=0}^{3} nOp_i \times LOp_i \tag{5.2}$$

In the presence of nested loops, L_{clk} is determined using Eq. 5.3, computed in a recursive manner, considering M the total number of loops.

$$L_{clk} = \sum_{i=1}^{M} II_i + (TC_i \times IL_i) \tag{5.3}$$

Where M is the number of nested loops, TC_i is the trip count corresponding to the $M-th$ level loop, and i represents the index loop.

Loop pipelining

In the presence of **perfect loops** (nested or single), the latency (L_{clk}) due to loop pipelining is defined as in Eq. 5.4, with j the level of the pipelined loop. In this case, the inner loop could be pipelined while the outer loops can be flattened (in absence of unrolling directive). Therefore, TC is the product between the different nested loops. Moreover, in the absence of the unrolling directive, $UF = 1$.

$$L_{clk} = D_j + II_j \times \left(\frac{TC}{UF_j} - 1 \right) \tag{5.4}$$

With $TC = \prod_{i=1}^{M} TC_i$, where M is the number of nested loops and TC_M is the trip count corresponding to the $M-th$ level loop.

Regarding **non-perfect loops**, the latency is obtained through the Eq. 5.5, as in [85], which considers the depth of the pipeline (D) in the overall computation, with j the level of the pipelined loop.

$$L_{clk} = D_j + II_j \times \left(\frac{TC_j}{UF_j} - 1 \right) \tag{5.5}$$

Loop unrolling

In the presence of **a single loop**, the latency (L_{clk}) due to loop unrolling is defined as in Eq. 5.6, according to [85].

$$L_{clk} = IL \times \frac{TC}{UF} \tag{5.6}$$

For **nested loops**, considering M the outermost loop, the iteration latency IL_{M-1} corresponding to the $M-1$ loops is defined by the Eq. 5.7 [85]:

$$L_{M-1} = \sum_{i=1}^{M-1} IL_i \times \frac{TC_i}{UF_i} \tag{5.7}$$

Including the outer loop, the total latency (L) for a nested loop is defined by the Eq. 5.8.

$$L_{clk} = IL_{M-1} + \frac{TC_M}{UF_M} \times IL_M \tag{5.8}$$

Dynamic indexation

The latency associated with dynamic indexation is considered when accessing an array in a non-contiguous region through a non-constant index. To this end, micro-benchmarks were generated to obtain the latency associated with the operations involved in the index generation to access to the memory position.

Load and storage operations

The clock cycles consumed by each load and storage operation are considered in the final computation of the latency. Moreover, if an array partition is applied, the impact on reading and writing an array is affected by the partition factor.

5.3.2 Resource model

The resource model is based on analytical and statistical techniques and aims to predict resource utilization, mainly associated with BRAM, DSP, LUT, and FF. The different components of the model are illustrated in Fig. 5.5. Area estimation is computed considering static hardware (DSP, BRAM), reconfigurable hardware (LUT, FF), directives applied (no directives, array partition, loop unrolling, loop pipelining).

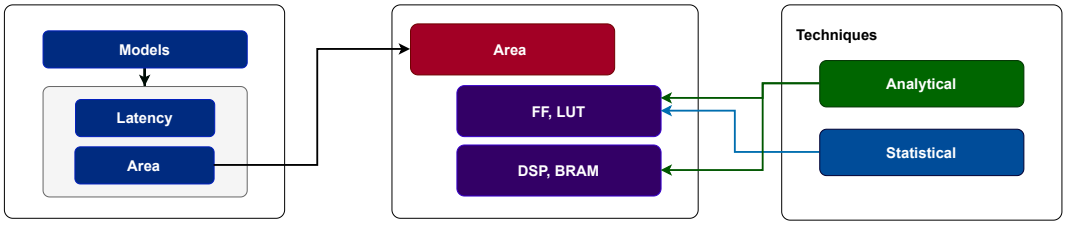


Figure 5.5: Resource utilization model.

LUT and FF estimation

The number of FF and LUT is affected by the number and type of operations involved in the arithmetic computation and by the generated expressions, multiplexer, and registers, among others.

Regarding the instance, which is mainly composed of the arithmetical operations performed, LUT and FF estimations are based on the premise that each elementary operation (addition, subtraction, multiplication, and division) is associated with the amount of LUT and FF through micro-benchmarks. Moreover, when the unroll directive is applied, the LUT and FF estimations are multiplied by UF for a given loop. The computation of LUT and FF for the instance, LUT_op and FF_op respectively, is depicted in Fig. 5.6. Once LUT_op and FF_op are computed, they are multiplied by UF.

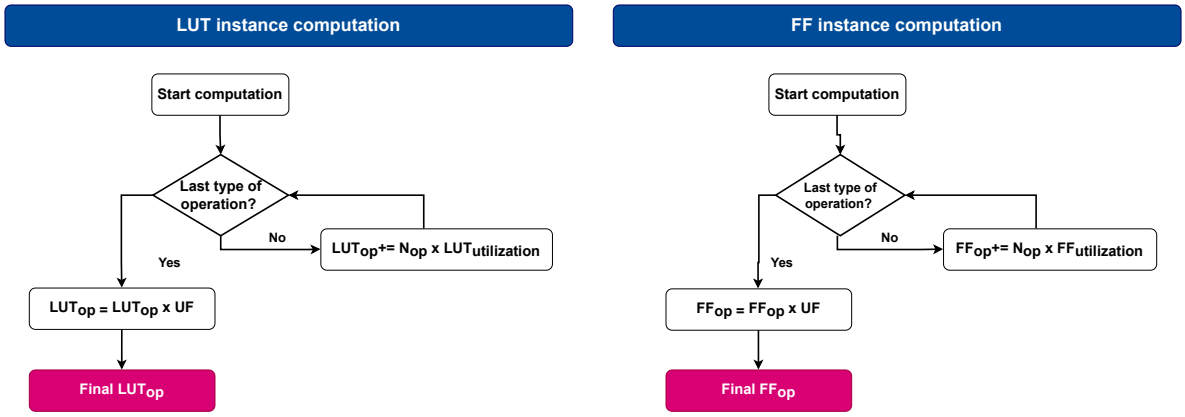


Figure 5.6: Flow chart for LUT and FF computation due to operations, corresponding to the instance's resources.

LUT due to expressions (LUT_{exp}), multiplexers (LUT_{mux}), and registers (LUT_{reg}) and FF due to expressions (FF_{exp}) and registers (FF_{reg}) utilization are obtained through multiple linear regression [212,213], a statistical technique that is used to predict the outcome of a variable based on the value of several explanatory variables.

In multiple linear regression, the regression model can be defined by the Eq. 5.9, where β_0 is the intercept, β_n are the partial regression coefficients, the number of observations $n = 1,2,...,k$, and $X_1, X_2, ..., X_n$ determine the explanatory variables [214].

$$Y = \beta_0 + \beta_1 \times X_1 + \beta_2 \times X_2 + ... + \beta_n \times X_n + \epsilon \tag{5.9}$$

The estimation of β_n is performed by the method of the least squares, extended to n dimensions. Details of multiple linear regression technique can be found in [215].

FF and LUT estimation is also affected by factors such as trip count, the number of operations, and the directives applied. Thus, several variables generate a final value for the different combinations. This approach aims to model the relations through multiple linear regression, avoiding modeling each of the different cases (expressions, multiplexers, and registers).

For the estimation of LUT utilization, and considering Table 5.2, the three independent variables (expression, multiplexer, and registers) obtained through multiple linear regression are associated with Nop, TC, Add, Mult, Sub, Div, NopExp from table 5.2. For FF utilization due to expression and registers, their value is obtained using the variables Nop, TC, Add, Mult, Sub, Div, NopExp

from table 5.2.

The number of LUT and FF consumed for the instance are computed based on the type of operations and the values presented in Table 5.1. The steps involved in the calculation due to multiple linear regression are presented in Fig. 5.7. First, it should be created the model corresponding to the regressor and perform the fitting with the data set (E.g., micro-benchmark from Table 5.2). Then, a prediction is performed for each variable to estimate (LUT_{exp}, LUT_{mux}, LUT_{reg}, FF_{exp}, and FF_{reg}).

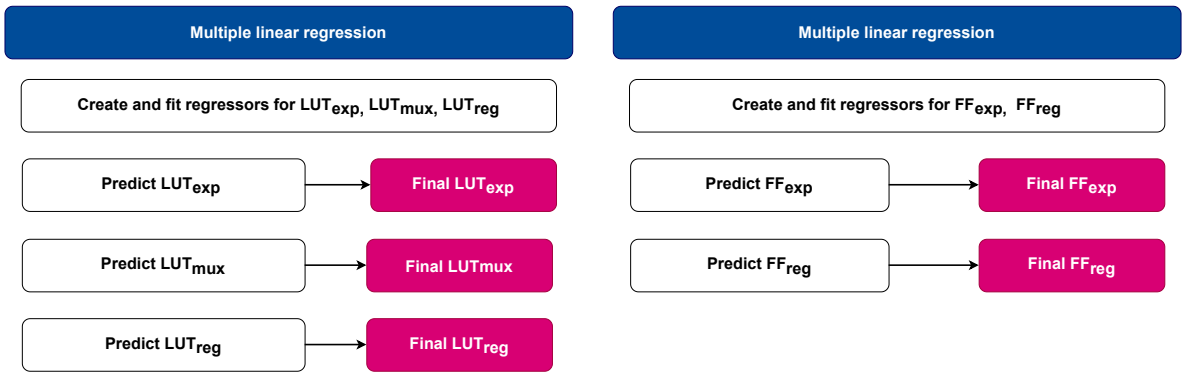


Figure 5.7: Steps involved in LUT and FF computation through multiple linear regression.

After the completion of the steps presented in 5.6 and 5.7, the estimated number of LUT and FF is obtained through Eq. 5.10 and 5.11, respectively.

$$LUT_f = LUT_{op} + LUT_{exp} + LUT_{mux} + LUT_{reg} \tag{5.10}$$

$$FF_f = FF_{op} + FF_{exp} + FF_{reg} \tag{5.11}$$

In MARTE, multiple linear regression was implemented through sklearn [216].

DSP estimation

The amount of DSP used for a specific computation depends on the type of operations executed. E.g., for a 32-bit floating-point, DSP utilization for each arithmetic operation is associated with the values presented in Table 5.1. Furthermore, if an unroll directive is used in a loop, the DSP utilization is also affected by UF, which means that the amount of DSP increases proportionally

with UF. The total amount of DSP (DSP_F) is obtained through Eq. 5.12, where Nop_{add}, Nop_{mult}, and Nop_{sub} represents the number of addition, multiplication, and subtraction operations respectively. DSP_{add}, DSP_{mult}, and DSP_{add} are the number of DSP consumed for each operation, and UF the unrolling factor.

$$DSP_F = \left\lceil \left(Nop_{add} \times DSP_{add} + Nop_{mult} \times DSP_{mult} + Nop_{sub} \times DSP_{sub}\right) \times UF \right\rceil \tag{5.12}$$

If the pipeline directive is applied for the estimation of DSP in the presence of resource sharing, the minimum number of instances for computation (N_r) is defined according to [85, 217]; thus, $N_r = \left\lceil \frac{N_{op}}{II} \right\rceil$, where N_{op} is the number of operations that are performed inside a loop.

BRAM estimation

BRAM estimation is associated with arrays that are stored in BRAM modules. For example, BRAM in UltraScale architecture-based devices stores up to 36 kb of data and can be configured as either two independent 18 kb RAMs or one 36 kb RAM. Each BRAM has two write and read ports [218].

To obtain the number of BRAM, for a given number of arrays in the source code, the algorithm iterates for each array, and based on the data type nB and number of words w (elements), if the result of $nB \times w \geq 1024$ an array will be stored in a block RAM.

To compute the amount of BRAM required by the algorithm, Eq. 5.13 expresses this relationship, where the number 8192 represents the 18 kb RAM and k represents the number of arrays in the applications. The final value is rounded to the next integer value. If $nB \times w \leq 1024$, the array is stored in FFs.

$$BRAM = \left\lceil \sum \frac{nB_k * w_k}{8192} \right\rceil \tag{5.13}$$

5.4 Design space explorer

In the literature, design space explorers based on black-box optimizations (BBO) are employed using HLS in the loop , being a time-consuming stage because of its synthesis time [71]. In MARTE DSE, the evaluation of the objective and constraint functions involve the execution of the hardware

acceleration performance model presented in Section 5.1, avoiding the execution of HLS tools in the loop, thus reducing the DSE engine runtime.

One of the main features of a DSE engine is the ability to predict an optimal (or suboptimal) combination of directives to be suggested to the user. To accomplish this with MARTE, the DSE engine comprises two sub-engines: single-objective Bayesian optimization and multi-objective optimization based on evolutionary algorithms (EA). The former optimizes latency. The latter optimizes both objective functions (area and latency) using EA.

Moreover, MARTE DSE prunes the exploration space, under hardware and user constraints, reducing the design points and helping to retrieve the feasible solutions, with less amount of points that are suitable for the cost model. Furthermore, a set of rules to guide the exploration of DSE engine are contemplated, as in [85, 127, 168].

In turn, when using FPGA-based devices, the first metric to be optimized is latency, taking advantage of the features of the architecture. Therefore, MARTE will explore the design space looking to optimize latency.

5.4.1 Single-objective Bayesian optimization

The sub-engine based on single-objective Bayesian optimization aims to **optimize latency**. Therefore, the objective function $f(x)$ is defined as the analytical model for estimating this metric presented in Section 5.1.

For single-objective optimization, the search space is defined as the combination of directives, their parameters, and user constraints. The surrogate function is based on a Gaussian process (GP) with the kernel defined as Matérn [219], one of the most common kernels for GP. The acquisition function $a(x)$ is defined as the expected improvement (EI) [220, 221], with which a reward equal to the improvement obtained is received, avoiding getting stuck in local optima. EI is employed to decide where to sample next.

Fig. 5.8 presents the flow of the single-objective based on BO. As the first step, the selection of the initial sample to evaluate can be performed in several manners: (i) HLS tool is executed to obtain a base estimation, and the generated report is loaded into MARTE through an XML parser, (ii) MARTE computes the first performance estimation without directives applied, (iii) randomly samples the search space to select the first configuration and execute MARTE to obtain the initial

estimation.

After the initialization step, the surrogate models the objective function with the Bayesian statistical model. If the stopping criterion is not met, then the search space sampling looks to maximize EI (acquisition function). This process implies selecting the following configuration to be evaluated through MARTE to obtain the latency estimation. Afterward, the posterior distribution is updated. This process continues until the search is stopped, retrieving the proposed configurations to the hardware developer.

The number of variables involved in the optimization process is defined according to the number and type of directives and their parameters (e.g., if pipelining is applied, *II* becomes a variable for the optimization problem). The variable types are integers because the parameters are represented by integer values. The principal constraint applied is that the maximum area occupied by a hardware design is at most 80% of programmable logic [170].

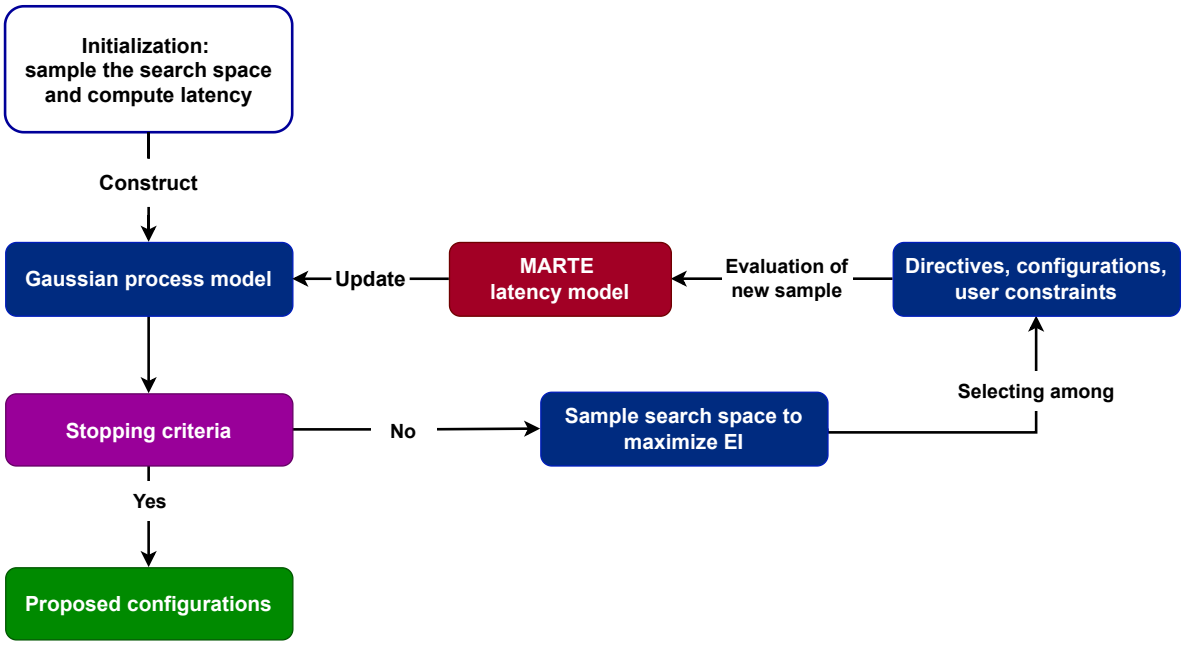


Figure 5.8: Flow diagram of the single-objective BO for DSE.

5.4.2 Evolutionary multi-objective optimization (EMO)

To implement a DSE engine using EMO, the problem to be solved is to optimize two objective functions: latency and area, that conflict with each other, which means that as latency decreases,

the area utilized increases.

The procedure for the implementation of the multi-objective optimization based on EA is presented in Fig. 5.9. The solver to drive the rule-based DSE is based on NSGA-II, that, as was exposed in Section 2.4.2, is an elitism algorithm that keeps the best solutions of the previous iterations, with a reduced complexity due to the non-sorting algorithm. The stopping criterion is fixed according to the maximum number of generations. Genetic operators, such as crossover, mutation, and selection, help to guide the search towards the solution, preventing the DSE engine from getting stuck in a local minimum.

Regarding the variables in the process of DSE based on EMO, the same considerations for DSE based on BO are taken into account.

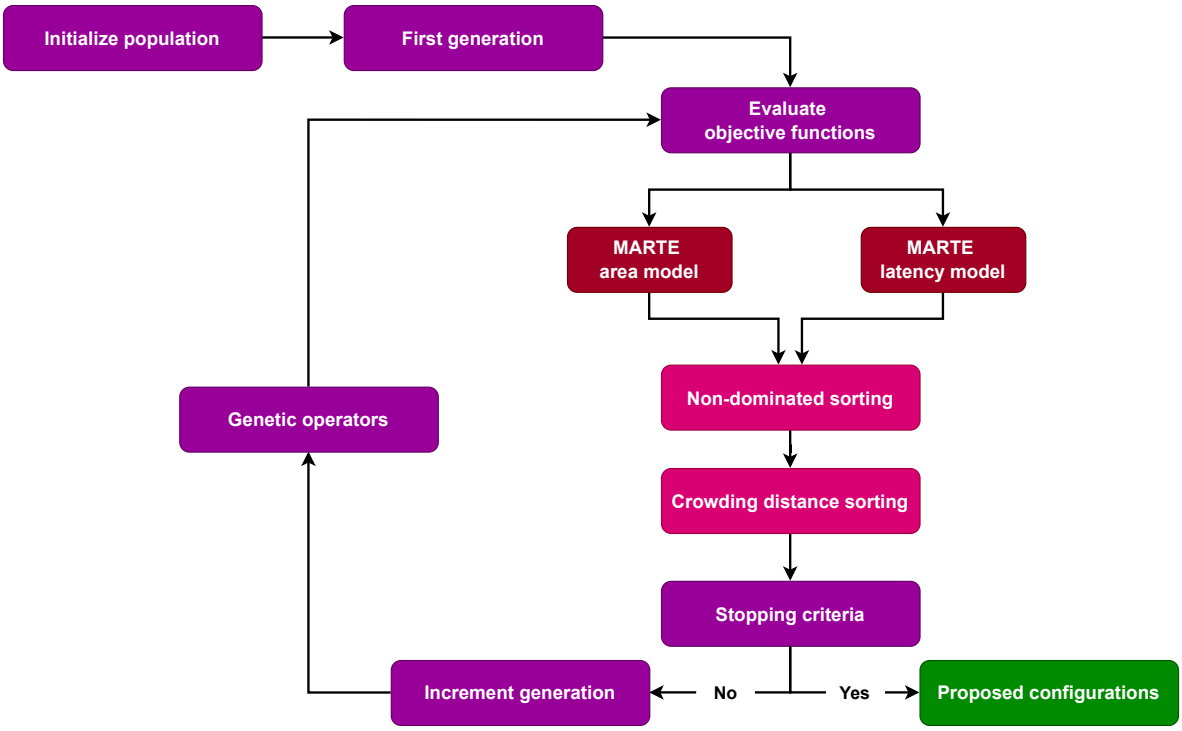


Figure 5.9: Flow diagram of the EMO for DSE with NSGA-II solver.

MARTE EMO DSE was implemented through Pymoo library [222], that is a framework for implementing state-of-the-art single- and multi-objective optimization algorithms.

5.4.3 Rules for guiding the DSE engine

According to [223], there are three types of constructs in the behavioral description that impact the final microarchitecture generated with HLS tools: loops, arrays, and functions. Moreover, three main exploration dimensions are available with HLS tools: local synthesis directives, number of functional units, and global synthesis options, which are orthogonal and can overlap. Based on this premise, exploration of the design space is constrained to loops, arrays, and functions under the influence of local synthesis directives.

HLS tools have automatic modes when a specific directive is applied [7, 9], which are used to build a set of rules to guide the exploration of the DSE engine, described as follows:

- In a nested loop, if the outer-loop is pipelined, the inner-loop is unrolled; if the top-level function is pipelined, all loops inside the functions must be unrolled.
- The optimal balance between area and performance is typically found by pipelining the innermost loop.
- Automatic flattening of the loop in absence of data dependencies.
- The maximum number of elements that can be partitioned completely is defined by the o HLS partitioning limit and tied to the tool version: 4,096 or 1,024 elements. If an array contains fewer elements than the maximum number of elements constrained by the vendor, an automatic array partition is considered.
- Some parameters are available depending on the applied directive. For example, if pipelining is applied, the unroll factor cannot be specified.

5.5 Outputs

In this last stage, the outputs of the performance estimator are reported to the hardware developer: design points after the exploration of the design space, area, latency, and a high-performance configuration suggestion.

As in the Roofline model, scalability *SC* of the system is defined as the ratio between the total resource available in the reconfigurable architecture and the maximum resource utilization by a

given PE, thus the *SC* of one PE is constrained by the most utilized resource, as it is shown in Eq. 5.14.

$$SC = \min\left\{\frac{BRAM_T}{BRAM_{PE}}, \frac{DSP_T}{DSP_{PE}}, \frac{LUT_T}{LUT_{PE}}, \frac{FF_T}{FF_{PE}}\right\} \tag{5.14}$$

5.6 Summary of the chapter

This chapter introduced MARTE, a comprehensive performance estimator for hardware acceleration composed of cost models and DSE engines.

The performance estimator is based on analytical and statistical techniques to compute latency and area (resource utilization), considering several modes in each case. The DSE engines are based on two independent sub-engines: single-objective based on BO and EA multi-objective optimization.

Bayesian optimization is one of the most used black-box optimization techniques to explore the design space, as presented in Section 4.1.2. For this reason, single-objective BO was paired with MARTE as DSE for latency optimization.

For optimizing latency and area, a DSE engine based on EMO is integrated, allowing a set of feasible solutions to assist the hardware developer when selecting the proper combination of directives with the area-latency trade-off. MARTE DSE engines consider a set of rules for guiding the exploration of the search space, reducing the points to be evaluated.

Furthermore, since one of the main drawbacks of evolutionary algorithms is that the search can stop at a local minimum, the single-objective optimization based on BO acts as a redundant algorithm to explore the search space, which increases the runtime. Furthermore, the different genetic operators allow a better convergence, trying to prevent the algorithm from remaining at a local minimum.

Chapter 6

Image analysis and highly demanding applications

This chapter presents the different cases of study considered in this thesis: pulse shape discriminator for cosmic rays in Section 6.2, automatic pest classification in Section 6.3, and re-ranking algorithm for information retrieval in Section 6.4. The applications are traversed for an ensemble of compression techniques to obtain a suitable implementation on FPGA/SoC, emphasizing the benefits when these types of applications are traversed by the compression techniques when targeting SoC-based FPGA devices.

6.1 Image analysis and other highly demanding applications

Applications in the field of image analysis are a relevant research focus in the scientific community [16–18]. The growth of artificial vision techniques for the processing, recognition, and classification of images has made it possible to expand the expectations of systems to solve problems that are otherwise much more difficult or impossible in different fields, such as security, industry, and autonomous driving.

As presented in [1], in recent years, ML techniques have been applied in multiple fields such as fluid dynamics, high-energy physics, information retrieval, image processing, video processing, security, and biology [19–21]. Because of this trend, models for FPGA-based architectures are being developed to accelerate ML applications with efficient exploitation of hardware resources to

improve productivity in the design phase [22–24].

This research considers three cases of study: pulse shape discriminator for cosmic rays, automatic pest classification, and re-ranking algorithm. The applications are traversed for an ensemble of compression techniques and targeting FPGA/SoC.

6.2 Pulse shape discrimination for cosmic rays

Cosmic rays consist of very high-energy particles that come from outer space. Primary cosmic rays may collide with other particles, splitting the molecules to form secondary cosmic ray particles. As a result of the collision, an air shower is created, containing secondary cosmic ray particles, such as neutrons, protons, positive and negative pions, and positive and negative kaons. Pions and kaons may decay into muons and neutrinos [226].

Data acquisition systems (DAQ) based on FPGAs and System-on-Chip are often used in experimental physics to perform data acquisition, and processing [51, 224, 225]. Experiments such as COMPASS at CERN [227] and the Latin American Giant Observatory (LAGO) [25] are examples where these devices are used as DAQ.

Water Cherenkov Detector (WCD) consists of a pure water tank used as a scintillator, coupled to a photomultiplier tube, connected to a high-voltage power supply, and an analog front-end [25], as presented in Fig. 6.1. The analog data from this setup are often digitized, captured, and processed within the detector electronics. However, some of the captured data traces may not be relevant and will have to be deleted in the subsequent offline data analysis.

In the context of WCD and cosmic rays, a WCD DAQ system has been utilized as a test base for deploying a neural network (NN) design within an existing project. The main goal was to verify advanced processing algorithms' versatility as co-existing processing blocks. The original system

Section 6.2 is based on the contributions published in [51, 224, 225]: [51] **Molina, R. S.**, Garcia, L. G., Morales, I. R., Crespo, M. L., Ramponi, G., Carrato, S., Cicuttin, A., Perez, H. (2022). "Compression of NN-Based Pulse-Shape Discriminators in Front-End Electronics for Particle Detection". In International Conference on Applications in Electronics Pervading Industry, Environment and Society, pp. 93-99. Springer, Cham. [224] **Molina, R. S.**, Crespo, M. L., Carrato, S., Ramponi, G., Cicuttin, A., Morales, I. R. Perez, H. (2021). "Muon–Electron Pulse Shape Discrimination for Water Cherenkov Detectors Based on FPGA/SoC". Electronics 2021, 10, 224. MDPI. [225] García Ordóñez, L. G., **Molina, R. S.**, Morales Argueta, I. R., Crespo, M. L., Cicuttin, A., Carrato, S., Ramponi, G., Pérez Figueroa, H. E., Ballina Escobar, M. G. (2021). "Pulse shape Discrimination for Online Data Acquisition in Water Cherenkov Detectors Based on FPGA/SoC". In 37th International Cosmic Ray Conference (ICRC2021), p. 274. PoS Sissa. In collaboration with the Abdus Salam International Centre for theoretical Physics (ICTP) - MLAB in Trieste Italy.

design has been paired with a NN to work as a trace/event discrimination block.

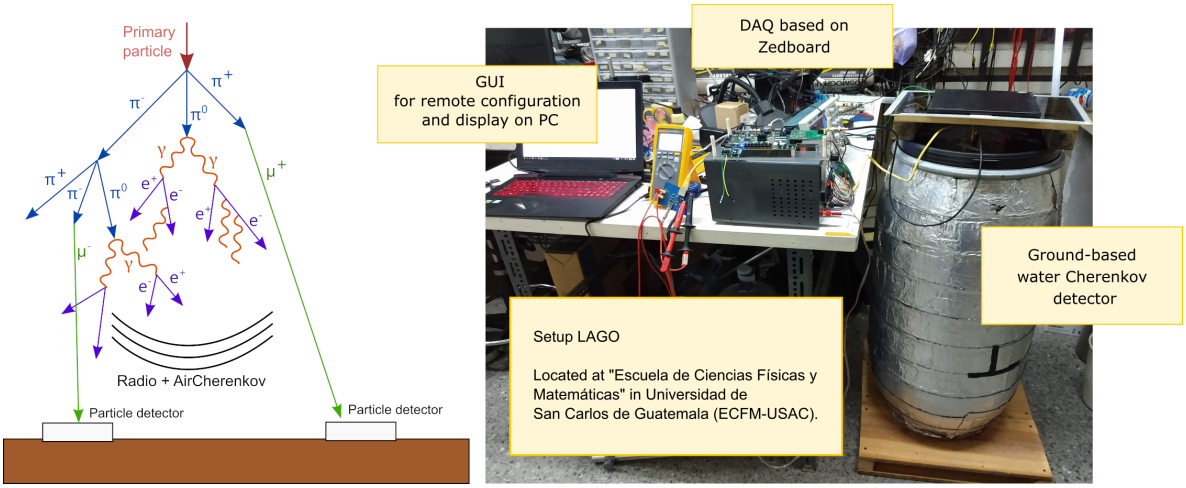
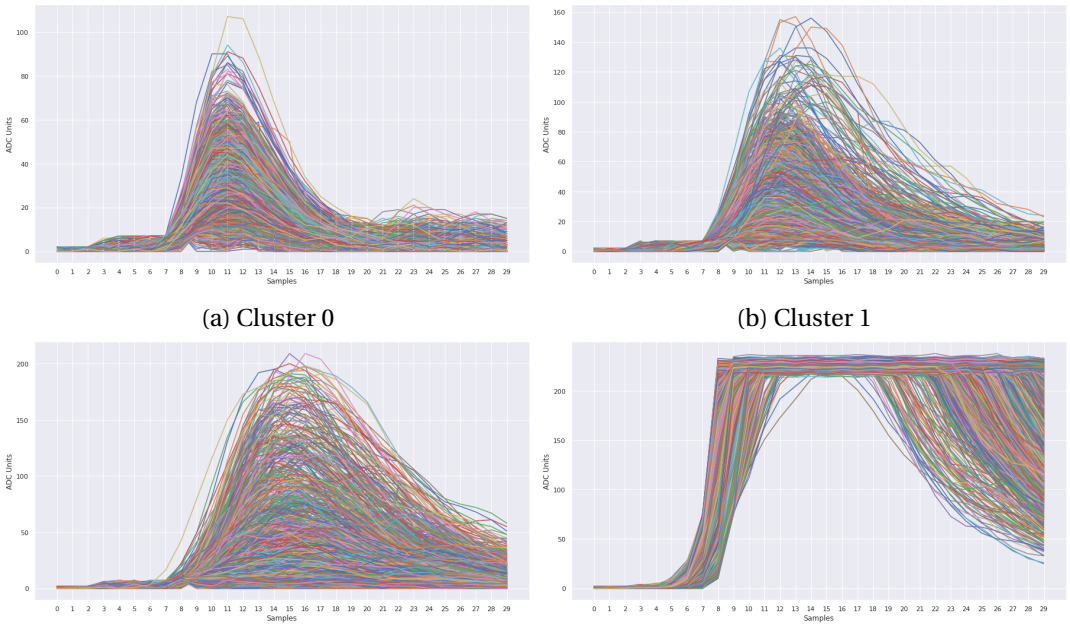


Figure 6.1: Cosmic rays DAQ system for Water Cherenkov Detectors.

In the literature, machine learning techniques have been used for offline pulse shape discrimination (PSD), obtaining high accuracy using floating-point precision [228–231]. From this starting point, performing a PSD within the FPGA/SoC based on ML-models is desirable.

6.2.1 Dataset

A set of ~3 million triggered pulses have been acquired with the DAQ system. An objective way of classifying the incoming signals is by grouping them by their similarities. For this case of study, such classification was done with a centroid-based clustering algorithm, an unsupervised machine learning technique that allows the dataset division into several groups called clusters. A distance metric is used to perform this classification in different groups based on similarity. If the distance between 2 elements is minimum, then both elements belong to the same cluster. Based on this, elements in the same cluster have similar features or properties. If the similarity measure is based on shape or pattern, the correlation as a distance measure is useful in this context [232], allowing the system to distinguish among pulse waveforms. The acquired pulses were classified using this method, and Fig. 6.2 presents a subset of raw pulses within each cluster of interest.



(a) Cluster 0

(b) Cluster 1

(c) Cluster 2

(d) Cluster 2a. (saturated)

Figure 6.2: Samples of raw pulse traces of each cluster. From [224].

6.2.2 ML-based architecture

This application's main objective is to classify four types of signals through an MLP architecture. As FPGA/SoC is the final platform for implementing PSD based on MLP, the model should be compressed to make efficient use of this technology, considering resource utilization and latency, without compromising the final performance of the overall system in terms of accuracy. In this direction, model compression was performed using the methodology presented in Section 2.2.

An MLP architecture that defines the teacher model is presented in Fig. 6.3, composed of four hidden layers with ReLu as the activation function and an output layer with four neurons with Softmax as the activation function. Through BO hyperparameter tuning, the number of neurons and the learning rate are obtained for each dense layer.

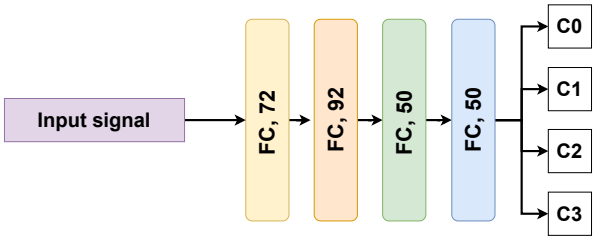


Figure 6.3: MLP teacher architecture.

After the BO hyperparameters tuning, the best configuration is selected. The network training is performed using early stopping callback to prevent the model from training with minimal benefits. The loss function is categorical cross-entropy, the batch size is 128, and the number of epochs is 32. The total number of parameters for the teacher architecture is 16,352, and the area under the curve (AUC) is higher than 0.95 for all the classes.

Several architectures with a precision of 32-bit floating-point precision are proposed as student networks, varying the number of hidden layers, as presented in Table 6.1. BO hyperparameters tuning is employed to obtain the number of neurons and kernels for each layer and the learning rate during the distillation process. Once the best configuration for the hyperparameters is found, the distillation of the student is performed.

Model	**Hidden layers**	**Parameters**	**CR**	**Configuration**
M1	3	529	30.91	I-15-O
M2	4	869	18.82	I-25-3-O
M3	5	731	22.37	I-15-13-3-O
M4	6	1623	10.08	I-35-13-3-3-O
M5	7	2245	7.28	I-45-13-3-3-3-O

Table 6.1: Distilled architectures based on MLP.

The verification of the effectiveness of the different networks is made by using the area under the curve (AUC) of receiver operating characteristic (ROC) [233], AUC-ROC, extended to multi-class classification by applying a one-vs-all technique. Various works in the literature support the use of

ROC for analyzing the behavior of machine learning classifiers [234,235]. Fig. 6.4 presents AUC per class for each student network.

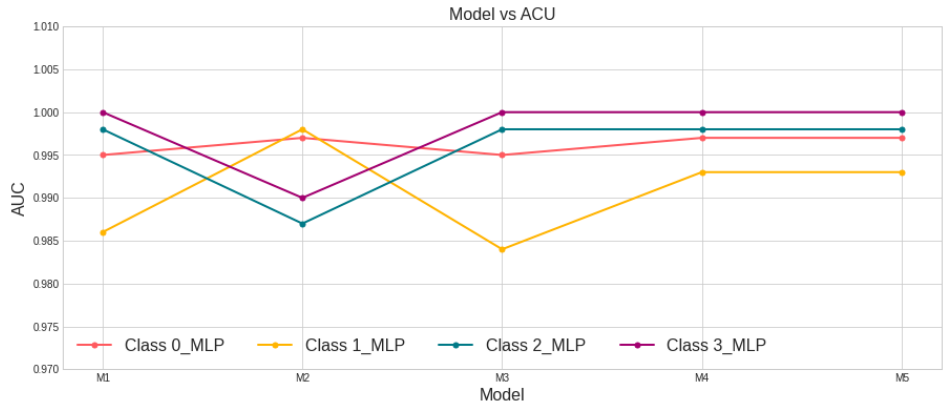


Figure 6.4: AUC per class for each student network.

6.2.3 ML-based model compression

After the student selection (M1 model), the next step is to perform the compression through quantization (4, 8, and 16 bits) and pruning (constant sparsity-based pruning 20%, 60%, and 80%). Moreover, quantization and pruning aware-training is combined with distillation (Distillation + constant sparsity-based pruning 20% + quantization 8 bits), considering the number of neurons and kernels obtained for the 32-bit floating-point distilled version with BO. Table 6.2 presents the result of this process.

$M1_{4b}$, $M1_{8b}$, and $M1_{16b}$ are models generated after quantization-aware training, considering 4, 8, and 16 bits fixed-point. As can be noticed with $M1_{4b}$ model, 4-bit fixed point showed a decrease in accuracy, affecting the final system's performance. Between models with 8 and 16 bits, the accuracy slightly drops from 95.46% to 93.2% for class 1. Considering the 3 models and the final device to deploy the model, 8-bit fixed-point is a good candidate to quantize the model.

Regarding pruning, increasing the percentage of sparsity leads to a slight decrease in accuracy. Nevertheless, considering the 20% 60%, and 80% of sparsity, the accuracy values are higher than 96% for the three models.

Therefore, selecting model $M1_{8b}$ to be combined with a sparsity factor of 20% ($M1\_8b_{P20}$), the accuracy remains higher than 95%, obtaining the benefits of both techniques.

$M1_{AQK}$ represents the model obtained through AutoQKeras, a functionality provided by QKeras to use optimization techniques (random grid, random search, Bayesian) to find an adequate number of bits for quantization. In this application, the result obtained for M1_8b_P20 is better than the one with AutoQKeras, considering the accuracy as a metric.

Finally, the last model, $M1_{DQP}$, was obtained by combining quantization (8-bit fixed-point), pruning (20% of sparsity), and knowledge distillation using a teacher architecture defined with 32-bit floating points. $M1_{DQP}$ has a good trade-off between the benefits of compression and the accuracy obtained for each class.

Model	**Parameters**	**AC c0**	**AC c1**	**AC c2**	**AC c3**
$M1_{4b}$	529	99.98	83.12	98.68	100
$M1_{8b}$	529	99.8	93.2	99.72	100
$M1_{16b}$	529	100	95.46	99.72	100
$M1\_32b_{P20}$	529	99.89	99.48	99.92	100
$M1\_32b_{P60}$	529	99,98	98.06	99.9	100
$M1\_32b_{P80}$	529	99.94	96.26	99.46	100
$M1\_8b_{P20}$	529	95.32	99.11	98.58	100
$M1_{AQK}$	242	99.66	84.01	98.88	99.9
$M1_{DQP}$	529	100	96.2	99.64	100

Table 6.2: Distilled architectures based on MLP. The four classes of pulse are represented by c0, c1, c2, and c3. AC stands for accuracy.

6.2.4 Implementation results

Once the compression process is completed, the next step is to generate the HLS project through the hls4ml [52] package. Different combinations based on the reuse factor (RF) for 1, 8, 16, 32, and 64 values are reported in Table 6.3 for two types of devices: ZCU102 and PYNQ-Z1. For KRIA platform, only the RF of 1 is presented.

As the sparsity is defined to be 20% of the total number of parameters, the total non-zero

weights determine the computational intensity and their impact on the hardware utilization. Thus, the operations that involve zero weights are not implemented with DSP.

In all the cases, the input and Softmax layers were implemented with 32-bit fixed-point, defined through hls4ml package before building the HLS project.

In order for hls4ml to be supported for Vitis HLS 2021.1.1, the source code was adapted, changing manually the directives used by the needed firmware.

Platform	BRAM	DSP	FF	LUT	Latency [clk]	RF	SC factor
ZCU102	3 (0%)	25 (0%)	1275 (0%)	12939 (0%)	14	1	21
ZCU102	2 (0%)	21 (0%)	1681 (0%)	13085 (4%)	22	8	20
ZCU102	2 (0%)	13 (0%)	2369 (0%)	13272 (4%)	23	16	20
ZCU102	2 (0%)	8 (0%)	2716 (0%)	13261 (4%)	28	32	20
ZCU102	2 (0%)	5 (0%)	2949 (0%)	13222 (4%)	31	64	20
PYNQ-Z1	3 (1%)	29 (13%)	9334 (8%)	12858 (24%)	32	1	4
PYNQ-Z1	2 (0%)	25 (11%)	7129 (6%)	12574 (23%)	36	8	3
PYNQ-Z1	2 (0%)	14 (6%)	6465 (6%)	12789 (24%)	38	16	3
PYNQ-Z1	2 (0%)	8 (3%)	6293 (6%)	12769 (24%)	42	32	3
PYNQ-Z1	2 (0%)	5 (2%)	6028 (5%)	12749 (23%)	44	64	3
KRIA	4 (0%)	27 (1%)	2866 (1%)	7809 (2%)	12	1	35

Table 6.3: HLS reports for M1 MLP @200MHz, without AXI interface. Latency in clock cycles. RF: reuse factor (configuration option in hls4ml). For ZCU102 and PYNQ-Z1, the reports were obtained with Vivado HLS 2019.2.1. For the KRIA device, the report was obtained from Vitis HLS 2021.1.1.

After exporting the corresponding IP core, and targetting KRIA device, the final resource utilization after place & route (P & R) for the inference task (without data transfer) is reported in Table 6.4, for a clock of 200 MHz.

Platform	BRAM	DSP	FF	LUT	RF
KRIA	4 (2.78%)	33 (2.64%)	504 (0.22%)	3007 (2.6%)	1

Table 6.4: Place & route report for the KRIA device. From Vivado 2021.1.1.

Considering uniform quantization for the whole network, Table 6.5 presents the results for resource utilization obtained from Vitis HLS and Vivado 2021.1.1, considering 8-bits fixed-point data type, targetting KRIA device.

Platform	BRAM	DSP	FF	LUT	RF
Vitis HLS	4 (2.78%)	4 (2.64%)	358 (2.4%)	6035 (5.05%)	1
Vivado (P & R)	2 (1.4%)	17 (1.36%)	64 (0.03%)	218 (0.19%)	1

Table 6.5: Metric estimations for KRIA device, obtained from Vivado 2021.1.1. P & R stands for place & route.

6.3 Automatic pest classification based on CNN

In agriculture, fruit crops are mainly affected by various diseases and pests during their growth. If the control is not timely, this will lead to a reduction in the soil or harvest loss. The accurate and effective control of insect pests is essential to help fruit growers improve fruit yield. One of the methods to solve this task is a manual approach, which is time-consuming and susceptible to errors. New trends in data processing have provided different ways to solve this problem, using automatic and efficient image recognition methods. Contributions in the literature have been presented to address this problem using machine learning techniques, obtaining high accuracy, and employing state-of-the-art neural networks [238–242].

Using embedded systems based on FPGA for IoT enables different measurements to be performed in parallel. Thus, logic design can collect and process information from the environment using different types of sensors (air temperature, solar radiation, soil temperature, among others) while processing complex algorithms based on machine learning (ML) and reducing power consumption. Owing to its flexibility in reconfiguration, this type of device has begun to be used in agriculture [243].

Section 6.3 is based on the works published in [236] and [237]: [236] Suárez, A., **Molina, R. S.**, Ramponi, G., Petrino, R., Bollati, L., Sequeiros, D. (2021, November). "Pest detection and classification to reduce pesticide use in fruit crops based on deep neural networks and image processing". In 2021 XIX Workshop on Information Processing and Control (RPIC), pp. 1-6. IEEE. [237] **Molina, R. S.**; Carrer. V.; Ballina; M., Crespo, M. L.; Bollati, L.; Sequeiro, D.; Marsi, S. and Ramponi, G. (2022) “ML-based classifier for precision agriculture on embedded systems”. In International Conference on Applications in Electronics Pervading Industry, Environment and Society [Accepted - Waiting for publication]. Collaboration with ICTP-MLAB, Nectras, and UNSL-LEIS.

6.3.1 ML-based architecture

When implementing ML-based models in embedded systems, the complexity and performance of the design are related to the number of required resources. Compression techniques are essential for deploying ML models on resource-constrained devices while maintaining efficiency and effectiveness [42]. Based on this, part of the methodology presented in Section 2.2 is employed to obtain the final DNN architecture to be deployed into resource-constrained devices, shown in Fig. 6.5.

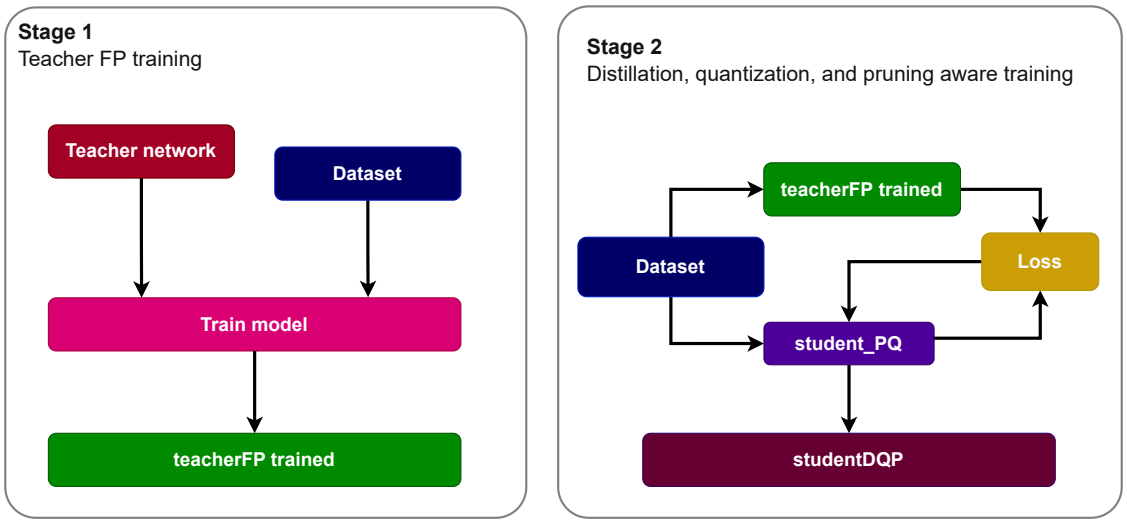


Figure 6.5: Methodology for compression.

Transfer learning technique [244] is used to obtain the teacher architecture, which is a binary classifier based on VGG16, presented in Fig. 6.6. The last layers of the model are replaced by a classifier based on [236]. The first four layers of this network are frozen, so they are not trainable, and the rest of the model will be fine-tuned. The final architecture is composed of 14,818,706 parameters.

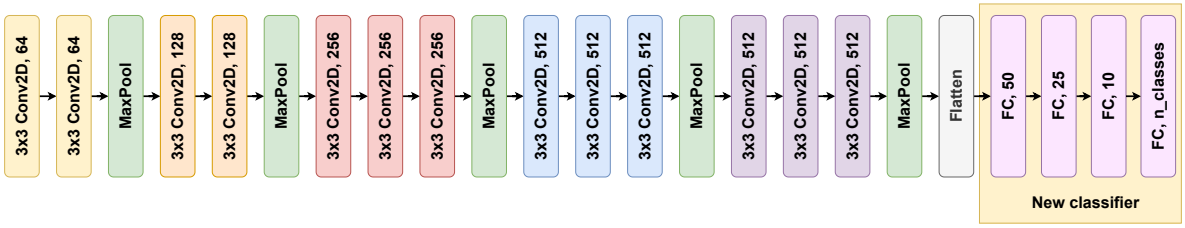


Figure 6.6: Teacher architecture based on VGG16, including the new classifier.

The teacher's knowledge (32-bits floating-point) is distilled to a student network, previously

defined with quantization and pruning strategies. The overall student network architecture (QStudentFPGA) is presented in Fig. 6.7, composed of 1,677 parameters and with the same number of layers as the teacher but with fewer kernels and neurons. The number of bits is 8 for the first four 2D-convolutional layers and 4 for the remaining layers. Finally, a target sparsity of 50% is applied to remove redundant parameters. Quantization- and pruning-aware training (QP-AT) is performed, which implies pruning, quantizing, and retraining the model to adjust and learn according to the newly quantized values and the distribution of neurons and connections.

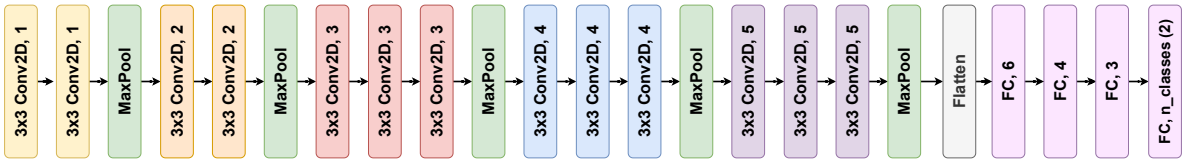


Figure 6.7: Distilled architecture.

6.3.2 Dataset

Several datasets for pest analysis in precision agriculture are available in the open literature. They differ in the source of the images and the captured insects: (i) *internet*, such as IP102 [245], which is based on the collection of different images obtained from the internet, (ii) *crop fields*, such as AgriPest [246], where the insects are captured directly in their habitat in different parts of the plant; and (iii) *traps*, such as Pest24, which is composed of images where the insects get stuck after being attracted, usually by pheromone lures. The selection of Pest24 for this research is motivated by the need for a particular purpose dataset for sticky paper traps.

Pest24 [247] is selected for fine-tuning the teacher and training the student networks. The dataset is divided into two classes for binary classification: Coleoptera (class 0) and Lepidoptera (class 1). Some samples are shown in Fig. 6.8. Regarding the inference process for the student networks, the test is performed with two different datasets composed of images from in-field traps: *Pest24* and a test set from the original application in Argentina (ARG dataset) provided by Nectras.

Figure 6.8: Samples of Pest24 dataset [247].

6.3.3 CNN assessment

The teacher network is obtained through **transfer learning** technique. The framework to implement the learning process is Keras, which allows the VGG-16 model to be loaded with the weights obtained from previous training with ImageNet. Regarding the hyperparameters, the Adam optimizer is used with a learning rate of 0.001, and the regularizer in each layer is L2 with a factor of 0.0001. The batch size is 32, as is the number of epochs. However, an early stopping mechanism is employed, which stops the training when the validation loss does not improve for five epochs (where a change of 0.005 counts as no improvement), then the weights of the model are restored when the loss starts to plateau. The confusion matrix for the teacher network is shown in Fig. 6.9. The final model size is 177.6Mb, making it challenging to implement in an embedded system. Thus, a compressed student network is generated through the knowledge distillation technique.

For the distillation process, the loss is configured as KLDivergence, Adam optimizer with 0.0001, batch size of 32, 128 epochs, an early stopping mechanism, and a callback monitor to reduce the learning rate when a specific metric has stopped improving after five epochs. The L2 regularizer is configured as in the teacher training. Pruning and quantization-aware training is performed.

Fig. 6.9 presents the confusion matrix for the student network (QStudentFPGA) tested on *Pest24* dataset, where high accuracy is noticed despite the compressed network.Nevertheless, the potential of transfer learning can be observed when working with datasets directly related to an application, as presented in Fig. 6.9 (QStudentFPGA ARG dataset). This could help in the early stages of the implementation of this system. Indeed, in-field fine-tuning is unnecessary if the reduced performances shown in matrix 3 in Fig. 6.9 are deemed acceptable.

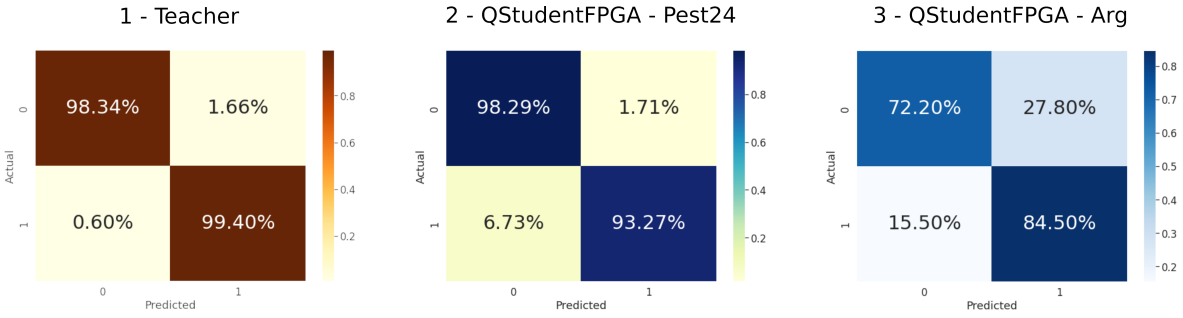


Figure 6.9: Confusion matrix: VGG-16-based teacher (1), QStudentFPGA *Pest24* dataset (2), QStudentFPGA ARG dataset (3).

6.3.4 Implementation results

For the SoC-based FPGA deployment, the memory footprint is measured in terms of hardware utilization (DSP, RAM, FF, and LUT). Therefore, the hls4ml package is employed to generate a high-level synthesis project to map the inference into the programmable logic. Once the IP core is created and integrated with the baseline hardware, the P & R reports provide the final resource utilization. For this purpose, we employed architectures based on FPGA: Xilinx PYNQ-Z1 and KRIA, considering a clock of 5ns. Table 6.6 presents the final resource utilization after P & R implementation, considering the whole hardware (inference IP core, processing system, DMA controller, AXI interconnect, and reset system). Moreover, table 6.7 presents the resource utilization after place and route implementation for the isolated inference IP core.

SoC-FPGA Fabric	BRAM	DSP	FF	LUT
KRIA	63 (22%)	20 (1.6%)	23920 (10.21%)	29772 (25.42%)
PYNQ-Z1	88 (63%)	26 (12%)	29441 (28%)	21543 (41%)

Table 6.6: Complete system. Utilization from P & R reports (post-implementation). Reports were obtained with Vivado 2021.1.1.

SoC-FPGA Fabric	BRAM	DSP	FF	LUT
KRIA	58 (20%)	20 (1.6%)	16297 (7%)	24513 (21%)
PYNQ-Z1	81 (63%)	26 (12%)	24886 (28%)	17937 (41%)

Table 6.7: Isolated inference IP core. Utilization from P & R reports (post-implementation). Reports were obtained with Vivado 2021.1.1.

6.4 Re-ranking algorithm

Section 6.4 is based on the works published in [248, 249]: [248] **Romina Molina**, Fernando Loor, Veronica Gil-Costa, Franco Maria Nardini, Raffaele Perego, and Salvatore Trani. 2021. Efficient traversal of decision tree ensembles with FPGAs. J. Parallel Distrib. Comput. 155, C (Sep 2021), 38–49; [249] Gil-Costa, V.; Loor, F.; **Molina, R.**; Nardini, F.; Perego, R.; Trani, S. (2022). Ensemble Model Compression for Fast and Energy-Efficient Ranking on FPGAs. In: Advances in Information Retrieval. ECIR 2022. Lecture Notes in Computer Science, vol 13185. Springer, Cham.. In collaboration with the Consiglio Nazionale delle Ricerche (CNR) - ISTI in Pisa.

6.4.1 Information retrieval system

Given an input query and a database composed of documents (millions or billions of elements), an information retrieval system is devoted to retrieve relevant documents for a given query. As depicted in Fig. 6.10, the information retrieval system comprises several steps: retrieval, initial ranking, re-ranking, and output of the final ranked documents.

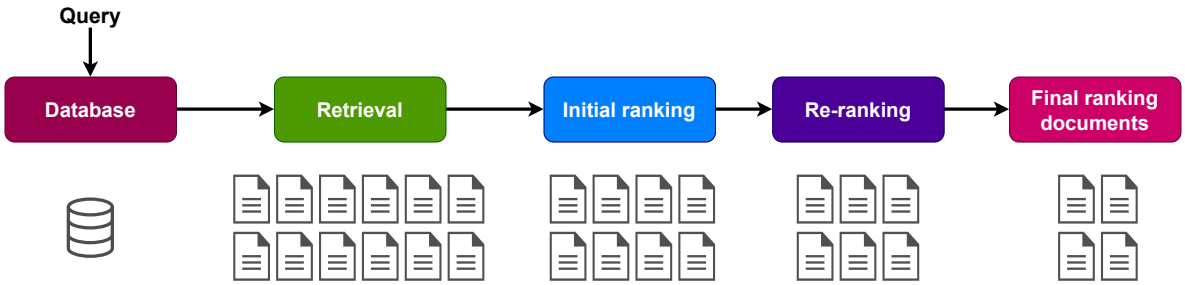


Figure 6.10: Information retrieval.

The retrieval stage aims to retrieve documents for a given user query and should be scalable and non-computationally expensive to facilitate data processing in subsequent stages. With a good set of relevant documents, an initial ranking is performed to determine the order in which the output will be presented to the user. In the final stage, a re-ranking operation of the candidates is performed to consider additional criteria or constraints. Finally, ranked documents are delivered to the user [250].

The information retrieval system is usually a part of web-scale search services deployed on cluster infrastructures and designed to support a peak request stream of thousands of queries per second. In this context, the database is distributed using redundancy strategies: each server holds a portion of the data, and the same partition is replicated on several servers to improve data availability and throughput and support fault tolerance. The results of each query are computed in parallel on all data partitions and are then merged and ranked for high precision using the ranking model [249].

6.4.2 Re-ranking through an ensemble of decision trees

The learned models to be accelerated are the additive *ensembles of decision trees*. These models are the most general and competitive solutions for several "difficult" tasks, such as ranking documents, items, or posts in Web search engines, e-Commerce platforms, or online social networks. They

are generated by boosting meta-algorithms that iteratively learn decision trees by incrementally optimizing a given loss function.

Owing to the incoming rate of requests and quality-of-service expectations, the traversal of such tree ensembles for many input instances must be fast and completed within small budgets. Because all these requirements are very challenging to fulfill, QS has been implemented on several architectures, such as manycore/multicore [251] and GPU-based implementaiton [252]. In addition, SoC-based FPGA architectures have shown their ability to accelerate intensive computing applications while saving power consumption, owing to their high parallelism and the architecture's reconfiguration capability. Therefore, the focus is on exploiting SoC-based FPGA features to efficiently deploy QUICKSCORER (QS), the state-of-the-art algorithm for the traversal of large tree ensembles [26, 27] constituting the *de facto* solution for the industrial deployment of complex ranking ML models (e.g., see [253–256]); considering, in the future, the implementation in AWS F1 instances.

The ML tree ensemble encompasses several binary decision trees, as illustrated in the rightmost part of Fig. 6.11. The internal nodes of each ensemble tree are associated with a Boolean test of the value of a specific feature that characterizes the input instance to be scored/predicted. Each leaf node stores a value representing the contribution of a specific tree to the final prediction.

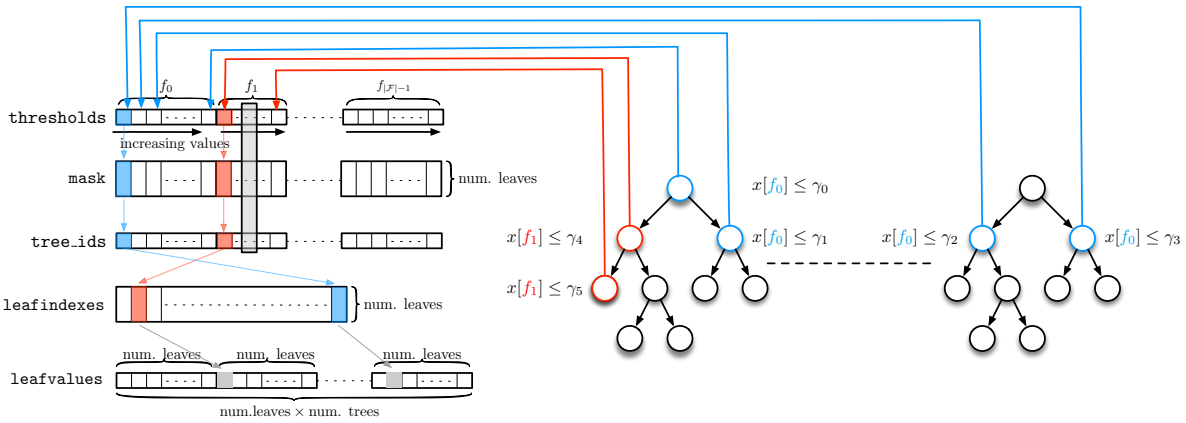


Figure 6.11: Data layout of the QS algorithm. From [248].

The QS algorithm 1 is composed by 3 main loops: mask initialization (MI_L), mask computation (MC_L), and score computation (SC_L). For details regarding QS algorithm, the reader can refer to [26, 27].

Algorithm 1: QUICKSCORER [26, 248]

Input :
- **x**: input feature vector
- $\mathcal{T}$: ensemble of binary decision trees, with
  - `thresholds`: sorted sublists of thresholds, one sublist per feature
  - `tree_ids`: tree's ids, one per internal split node
  - `mask`: node bitvectors, one per internal split node
  - `offsets`: offsets of the blocks of triples
  - `leafindexes`: result bitvectors of size Λ, one per each tree
  - `leafvalues`: output values, one per each tree leaf

Output:
- Final score of **x**

```
1  QUICKSCORER(x, T):
2      foreach t ∈ 0, 1, ..., |T| − 1 do                          // Mask Initialization (MI_L)
3          leafindexes[t] ← 11...11
4      foreach f ∈ 0, 1, ..., |F| − 1 do                          // Mask Computation (MC_L)
5          i ← offsets[f]
6          end ← offsets[f + 1]
7          while x[f] > thresholds[i] do
8              t ← tree_ids[i]
9              leafindexes[t] ← leafindexes[t] ∧ mask[i]
10             i ← i + 1
11             if i ≥ end then
12                 break
13     score ← 0
14     foreach t ∈ 0, 1, ..., |T| − 1 do                          // Score Computation (SC_L)
15         j ← index of leftmost bit set to 1 of leafindexes[t]
16         l ← t · Λ + j
17         score ← score + leafvalues[l]
18     return score
```

6.4.3 Towards the hardware implementation

The first QS hardware implementation was performed using a 32-bit floating point to obtain the performance estimation of the baseline application. To this end, the algorithm was ported to HLS tools (version 2019.1.1) by adding adequate directives and targeting the xczu9eg-ffvb1156-2-e FPGA with a clock frequency of 200 MHz. To perform the validation of the created hardware, a publicly available LtR dataset was used, namely MSLR-WEB30K-F1 (Fold 1)[1] [257], hereinafter abbreviated as MSN30K. The results of the hardware implementations are presented in Table 6.8, for 100 and 1000 number of trees. Due to the dynamic loop bounds of the internal loop in MC_L, HLS cannot accurately estimate the latency. Thus, a directive is added to force the tool to estimate the maximum number of iterations. It can be seen how, as the number of trees increases, so does latency.

[1] `http://research.microsoft.com/en-us/projects/mslr/`

	HLS report					
HD	BRAM	DSP	FF	LUT	Latency min.	Latency max.
A.	19	2	497	697	2181 [10.9 μs]	64481 [0.32 ms]
Utilization(%)	1.04	0.079	0.09	0.25	-	-
B.	343	3	1619	1563	8348 [41.74 μs]	1568348 [7.84 ms]
Utilization(%)	18.8	0.12	0.29	0.57	-	-

Table 6.8: Re-ranking algorithm - 32-bits floating-point version. Latency in clock cycles. The acronyms used in the table are: HD: Hardware design. [A]QS 100 Trees. No directives. [B]. QS 1000 Trees. No directives. Reports obtained through Vivado HLS 2019.1.1.

As noticed, to have an improvement in latency, some strategies based on code restructuring and memory footprint reduction should be implemented.

Binning and quantization strategies were used to reduce the memory occupation of ensemble models on FPGA-based devices [249]. A 32-bit real value is stored in each leaf with a data type of 8-bit unsigned integer for quantization, reducing 3/4 the space for storing the leaves of the ensemble.

As presented in [249], Fig. 6.12 shows the performance of the QS algorithm on the efficiency of the scoring process through NDCG metric [258]. Two versions of QS were implemented: the original version [26] and a new version supporting binned and quantized models. Both techniques devoted to reducing the memory footprint do not significantly affect the ranking effectiveness.

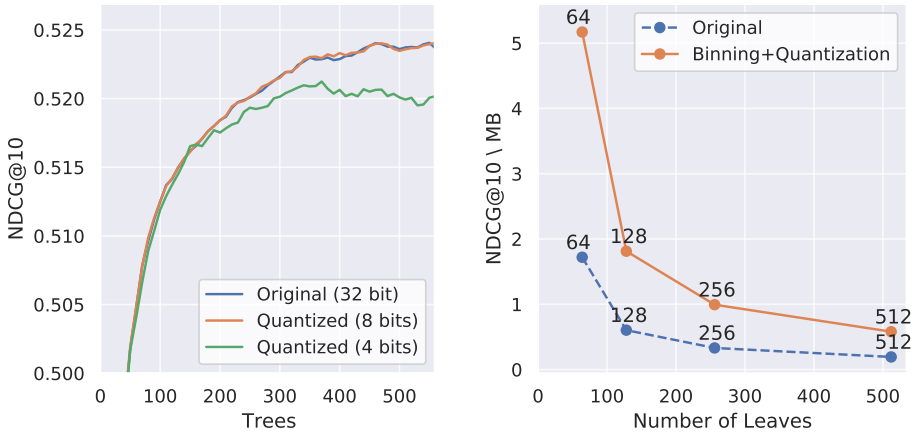


Figure 6.12: λ-MART efficiency/effectiveness trade-off. Right: NDCG@10 per MB of model size. Left: impact of quantization on NDCG@10. From [249].

The code restructuring technique is performed using several strategies. The MI_L is replaced by a data structure previously initialized. Then, the *while-loop* inside MC_L loop is transformed into a *for-loop* with variable bounds. Since HLS needs to know the size of the loop boundaries at compile time, two versions are proposed: (i) MC_L with fixed lower and upper bounds, (ii) MC_L is divided into four loops with fixed lower and upper bounds. Finally, MC_L and SC_L are divided into two different functions. The high-level representation of the IP core based on QS is presented in Fig. 6.13.

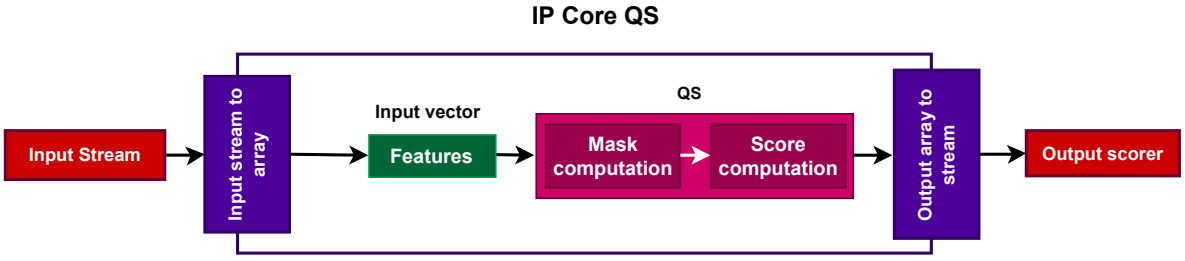


Figure 6.13: High-level representation of the QS IP core.

Table 6.9 presents the synthesis reports obtained using the HLS tools for the QS algorithm. From [A] to [D] QS binned and quantized versions, and from [E] to [F] 32-bit floating-point implementations. The best result is obtained with the hardware design [D] in terms of latency for a clock @ 200 MHz. The BRAM for the model storage was reduced from 377 to 198, revealing the impact of quantization. [D] has the loop MC_L divided by a factor of four, where each loop has a

PIPELINE II=1 directive, and array partitioning applied to the arrays input, output, tree_ids, thresholds, masks, and feature_remap (presented in Fig. 6.11). These directives are the same applied to the hardware design [F].

	HLS report					
HD	BRAM	DSP	FF	LUT	Latency min.	Latency max.
A.	188	2	1546	1647	10584 [52.9 μs]	3130584 [15.6 ms]
B.	188	0	1150	1621	2920 [14.6 μs]	1080992 [5.4 ms]
C.	188	0	499	1192	2784 [13.9 μs]	1091464 [5.5 ms]
D.	198	4	80252	39176	**974** [**4.9** μs]	**282494** [**1.42** ms]
E.	377	7	76406	38057	6569 [32.84 μs]	5584065 [27.9 ms]
F.	377	7	84937	44634	4895 [24.5 μs]	288319 [1.44 ms]

Table 6.9: Re-ranking algorithm. From [A] to [D] with binning and quantization strategies; [E] to [F] 32 floating-point version. Latency in clock cycles. The acronyms used in the table are: HD: Hardware design. [A]. No directives. Base implementation. [B]. No directives. One fixed loop [C]. No directives. Loop MC_L divided by a factor of four. [D]. Same case as [C], but with directives applied. [E]. Floating-point with code restructuring. No directives. [F]. Floating-point with Loop MC_L divided by a factor of four and directives.

6.5 Summary of the chapter

This chapter exposed three cases of study for image analysis and other highly demanding applications. It was observed how ad-hoc techniques, such as compression and machine learning, can lead to optimal hardware implementation with a good compromise between efficiency, effectiveness, memory footprint, and inference time. For all cases of study, a fully-on-chip deployment was achieved.

The pulse shape discriminator was implemented through an MPL architecture, using the methodology presented in Section 2.2 to obtain an efficient classifier. This case study aimed to perform a pulse shape discriminator in an electronic front-end for particle detection.

A CNN-based classifier for precision agriculture was developed for automatic pest classification, exploiting compression techniques to reduce the application's memory footprint. Moreover,

the potential of transfer learning when working with a dataset directly related to the application was exposed, thereby facilitating the introduction of ML-based classifiers during the early stages of implementing this type of system in the field. Nevertheless, changes in the external conditions can impact the input images, leading to accuracy degradation over time.

Re-ranking algorithm for information retrieval was implemented, taking into account 32-bit floating-point and 8-bit fixed-point data structures. Moreover, this case of study exhibited the effect of code restructuring techniques when targeting SoC-based FPGA, showing their benefits when targetting SoC-based FPGA devices. Moreover, this application is suitable for its acceleration through high-end FPGAs and AWS-F1 instances.

The three applications proved the benefits of compression techniques when targetting SoC-based FPGA to improve memory footprint and inference time.

The next chapter will present MARTE performance assessment.

Chapter 7

Experiments and results

This chapter presents the experiments and results related to the evaluation of MARTE, which is obtained through assessing the cost models, DSE engine, runtime, and portability. Section 7.2 presents the metrics used to assess the overall performance, Section 7.4 discusses the performance of the cost models for area and latency, considering basic (multiplication, matrix multiplication, and FIR filter) and highly demanding applications (PSD based on MLP, CNN-based pest classification, and re-ranking). Section 7.5 exposes the results of the MARTE DSE engine. Runtime and compatibility are shown in Section 7.6 and 7.7, respectively.

7.1 Experimental setup

The experimental setup is composed of:

- CPU1: Intel Core i7 3.4GHZ 64GB RAM, GeForce GTX 1070.
- CPU2: Intel Core i7 9750H, 24GB RAM, GeForce GTX 1050.
- High-level synthesis tools: Vivado 2019.1.1 and Vitis 2021.1.1
- SoC: Kria KV260 Vision AI Starter Kit and ZCU102.
- Libraries: Python 3.9, hls4ml 0.6, QKeras 0.9, Keras-tuner 1.1.2, Keras 2.9, TensorFlow 2.4.1, Pymoo 0.6.

7.2 Metrics

When studying a system through a macro approach, the global behavior of the application and its performance are measured. Based on this, to achieve MARTE performance evaluation, the following metrics are employed:

- **Prediction error (P_{error} [%]):** Because the total amount of each resource presented in the FPGA (FF, LUT, DSP, and BRAM) differs from each other, the prediction error P_{error} is measured as the absolute difference between the relative error corresponding to each resource computed with the HLS tool R_{HLS} and MARTE R_{MARTE}. This relation is presented in Eq. 7.1. RT_R represents the total amount available for a given resource R.

$$P_{error} = \left| \frac{R_{HLS} \times 100}{RT_R} - \frac{R_{MARTE} \times 100}{RT_R} \right| \tag{7.1}$$

  In this Chapter, RT_R represents the available resources on KRIA board, presented in Table 7.1, with R corresponding to a specific resource: BRAM, DSP, LUT, or FF.

Kria KV260 - Available resources			
BRAM	DSP	FF	LUT
288	1248	234240	117120

Table 7.1: Available resources on Kria KV260 development board.

- **Absolute difference (AD_L [clk]):** Defined as the absolute difference between the latency reported by HLS tool and MARTE, as presented in Eq. 7.2 and measured in terms of clock cycles [clk].

$$AD_L = |L_{HLS} - L_{MARTE}| \tag{7.2}$$

- **Latency ratio (L_{ratio}):** is measured as the relation between the latency computed with the HLS tool (L_{HLS}) and the latency obtained with MARTE (L_{MARTE}), as presented in the Eq. 7.3.

$$L_{ratio} = \frac{L_{HLS}}{L_{MARTE}} \tag{7.3}$$

- **Running metric:** allows observing the convergence of the system for EMO implementation, showing the difference in the objective space between the different generations, and it is used to evaluate the performance of the non-dominant solution set when the Pareto front is unknown [259].

7.3 Basic applications

The pseudo-codes corresponding to the basic applications are presented in Algorithms 2, 3, and 4 for multiplication, matrix multiplication, and FIR filter, respectively, to assist the explanation of the results presented in this Chapter.

Algorithm 2: Multiplication

Input :

- **a**: input vector 1
- **b**: input vector 2

Output:

- **c**: Final multiplication value between vectors **a** and **b**

```
1 Multiplication(a, b):
2     foreach i ∈ 0, 1, ..., |N| − 1 do                    // Multiplication
3         c[i] ← a[i] ∗ b[i]
4     return c
```

Algorithm 3: Matrix multiplication

Input :

- **A**[**MAT_A_ROWS**][**MAT_A_COLS**]: input matrix 1
- **B**[**MAT_B_ROWS**][**MAT_B_COLS**]: input matrix 2

Output:

- **C**[**MAT_A_ROWS**][**MAT_A_COLS**]: Output matrix

```
1 Matrix multiplication(A, B, C):
2     foreach i ∈ 0, 1, ..., |MAT_A_ROWS| − 1 do                    // Row
3         foreach j ∈ 0, 1, ..., |MAT_B_COLS| − 1 do                // Col
4             C[i][j] ← 0
5             foreach k ∈ 0, 1, ..., |MAT_B_ROWS| − 1 do            // Product
6                 C[i][j] ← C[i][j] + A[i][k] × B[k][j]
7     return C
```

Algorithm 4: FIR filter

Input :
- **x**: input signal
- **c**[**N**]: input coefficients

Output:
- ∗**y**: Output signal

```
1 shif_reg[N]
2 acc ← 0
3 data ← 0
4 FIR filter a, b, c:
5     foreach i ∈ N − 1,...,0 do                          // Shift_Accum_Loop
6         if i = 0 then
              shift_reg[0] ← x
              data ← x
7         else
              shift_reg[0] ← shift_reg[i-1]
              data ← shif_reg[i]
          acc ← acc + acc × c[i]
8     *y ← acc;
```

7.4 MARTE performance evaluation

MARTE performance assessment is presented, considering the models for latency and area. The results were obtained for the basic applications (multiplication, matrix multiplication, and FIR filter) and highly demanding applications (pulse shape discriminator, automatic pest classification, and re-ranking), showing the comparison between the HLS tool and MARTE.

7.4.1 Analytical models for resource and latency estimation

This section presents the metric estimations obtained using MARTE and HLS tools for the basic applications: multiplication, matrix multiplication, and finite impulse response (FIR) filter. It is considered a precision of 32-bit floating point for data structures and manual setting of directives.

Tables 7.2, 7.4, and 7.6 present a comparison between the metric estimation using the HLS tool and MARTE for DSP, BRAM, FF, LUT, and latency.

Multiplication

	HLS tool					Model				
HD	BRAM	DSP	FF	LUT	Latency	BRAM	DSP	FF	LUT	Latency
A.	0	5	238	202	51	0	5	205	196	51
B.	0	30	1282	1364	2	0	30	1324	1411	2
C.	0	3	305	228	15	0	3	226	207	13

Table 7.2: Metric estimation through HLS tool and MARTE for multiplication. The acronyms used in the table are: HD: Hardware design. A. No directives. [B] Unroll and array partition complete. [C] Pipeline II=1.

The corresponding P_{error}, AD_L, and L_{ratio} are shown in Tables 7.3, 7.5, and 7.7. In these three cases, the P_{error} for the resource utilization was below 1% and the L_{ratio} is lower than 2 for the three applications.

	P_{error} [%]				AD_L [clk]	L_{ratio}
HD	BRAM	DSP	FF	LUT		
A.	0	0	0.014	0.005	0	1
B.	0	0	0.03	0.04	0	1
C.	0	0	0.033	0.018	2	1.15

Table 7.3: P_{error}, AD_L, and L_{ratio} for multiplication. The acronyms used in the table are: HD: Hardware design. [A]. No directives. [B] Unroll and array partition complete. [C] Pipeline II=1.

Matrix multiplication

	HLS tool					Model				
HD	BRAM	DSP	FF	LUT	Latency	BRAM	DSP	FF	LUT	Latency
A.	0	5	516	551	241	0	5	456	717	240
B.	0	5	931	677	153	0	5	1083	691	154
C.	0	5	1351	1435	44	0	5	1308	1418	48
D.	0	15	2883	2212	27	0	15	1803	1892	27
E.	0	15	2206	1959	28	0	15	1748	1896	27

Table 7.4: Metric estimation through HLS tool and MARTE for matrix multiplication. The acronyms used in the table are: HD: Hardware design. [A]. No directives. [B] AP + Product Loop: Pipeline II=3. [C] AP + Col Loop: Pipeline II=3. [D]. AP + Row Loop: Unroll Factor=2 [E]. AP + Row Loop: Pipeline Factor=3.

	P_{error} [%]				AD_L [clk]	L_{ratio}
HD	BRAM	DSP	FF	LUT		
A.	0	0	0.026	0.142	0.415	1.004
B.	0	0	0.065	0.012	0.654	0.994
C.	0	0	0.018	0.015	9.091	0.917
D.	0	0	0.461	0.273	0.000	1.000
E.	0	0	0.196	0.054	3.571	1.037

Table 7.5: P_{error}, AD_L, L_{ratio} and for matrix multiplication. The acronyms used in the table are: HD: Hardware design. [A]. No directives. [B] AP + Product Loop: Pipeline II=3. [C] AP + Col Loop: Pipeline II=3. [D]. AP + Row Loop: Unroll Factor=2 [E]. AP + Row Loop: Pipeline Factor=3.

FIR filter

	HLS tool					Model				
HD	BRAM	DSP	FF	LUT	Latency	BRAM	DSP	FF	LUT	Latency
A.	0	5	576	526	100	0	5	456	415	100
B.	0	5	667	578	40	0	5	540	419	40
C.	0	5	571	557	60	0	4	540	419	60
D.	0	5	913	910	36	0	5	890	870	35

Table 7.6: Metric estimation through HLS tool and MARTE for FIR filter. The acronyms used in the table are: HD: Hardware design. [A]. No directives. [B]. Pipeline II = 3, [C]. Pipeline II = 5, [D]. Pipeline II = 3 + AP complete.

	P_{error} [%]				AD_L [clk]	L_{ratio}
HD	BRAM	DSP	FF	LUT		
A.	0.000	0.000	0.003	0.000	1.000	0.990
B.	0.000	0.000	0.051	0.095	0.000	1.000
C.	0.000	0.08	0.054	0.136	0.000	1.000
D.	0.000	0.000	0.010	0.034	0.107	1.029

Table 7.7: P_{error}, AD_L, and L_{ratio} for FIR filter. The acronyms used in the table are: HD: Hardware design. [A]. No directives. [B]. Pipeline II = 3, [C]. Pipeline II = 5, [D]. Pipeline II = 3 + AP complete.

7.4.2 Pulse shape discriminator: performance estimation

The implementation of the MLP architecture is performed using the hls4ml package to obtain the HLS project, from which the corresponding IP core is generated to map the inference phase to the FPGA. An 8-bit fixed point is used for the weights and bias of the MLP architecture. Because in hls4ml the directives are fixed, a reuse factor (RF) of 1 was considered in this implementation.

Table 7.8 presents a comparison between metric estimation using the HLS tool and MARTE for

DSP, BRAM, FF, and LUT. As presented in Table 7.9, the prediction error was below 2% for the FF and LUTs, with a latency ratio of 1.1.

	HLS tool					**Model**				
HD	BRAM	DSP	FF	LUT	Latency	BRAM	DSP	FF	LUT	Latency
A.	4	4	358	6035	10	4	4	739	7591	9

Table 7.8: Metric estimation through HLS tool and MARTE for pulse shape discriminator. The acronyms used in the table are: HD: Hardware design. [A]. 8-bits fixed-point, reuse factor = 1.

	P_{error} [%]				AD_L [clk]	L_{ratio}
HD	BRAM	DSP	FF	LUT		
A.	0.000	0.000	0.163	1.328	1	1.1

Table 7.9: P_{error} , AD_L, and L_{ratio} for pulse shape discriminator. The acronyms used in the table are: HD: Hardware design. [A]. Reuse factor = 1,

Considering the resource utilization reported after place and route, Table 7.10 exhibits the prediction error, showing that the highest value for LUT estimation, with a P_{error} of 6.3%.

	BRAM	**DSP**	**FF**	**LUT**
Vivado (P & R)	2	17	64	218
MARTE	4	4	739	7591
P_{error} [%]	0.69	1.04	0.283	6.3

Table 7.10: Pulse shape discriminator. P_{error} considering MARTE estimation and place and route report (Vivado 2021.1.1).

7.4.3 Automatic pest classification: performance estimation

In this implementation, 8-bit fixed point is considered for the weights and bias of the first two layers of the CNN architecture, while the remainder layers are defined with 4-bit fixed point. Table

7.11 presents a comparison between metric estimation using the HLS tool and MARTE for DSP, BRAM, FF, and LUT, considering the manual setting of directives. As presented in Table 7.12, the prediction error was below 1% for the DSP and FFs, while for BRAM and LUT was below 50%, with a latency ratio of 0.92.

	HLS tool					**Model**				
HD	BRAM	DSP	FF	LUT	Latency	BRAM	DSP	FF	LUT	Latency
A.	208	16	30480	71536	13477	64	21	29413	24582	14594

Table 7.11: Metric estimation through HLS tool and MARTE for automatic pest classification algorithm. The acronyms used in the table are: HD: Hardware design. [A]. Reuse factor = 1.

	P_{error} [%]				AD_L [clk]	L_{ratio}
HD	BRAM	DSP	FF	LUT		
A.	49.31	0.401	0.58	41.381	1117	0.92

Table 7.12: P_{error} , AD_L, and L_{ratio} for pest classification algorithm. The acronyms used in the table are: HD: Hardware design. [A]. Reuse factor = 1.

Nevertheless, taking into consideration Table 6.7 from Section 6.3, it can be observed the lower prediction error compared with the P & R results for KRIA device, exposed in Table 7.13.

	BRAM	**DSP**	**FF**	**LUT**
Vivado (P & R)	58	20	16297	24513
MARTE	64	21	29413	24582
P_{error} [%]	2.083	0.08	5.6	0.06

Table 7.13: Automatic pest classification. P_{error} considering MARTE estimation and P & R report (Vivado 2021.1.1).

7.4.4 Re-ranking algorithm: performance estimation

In this implementation, a 32-bit floating point is considered for the data structures for a re-ranking algorithm composed by an ensemble of 100 trees. Table 7.14 presents a comparison between metric estimation using the HLS tool and MARTE for DSP, BRAM, FF, and LUT, considering a manual setting of directives. As presented in Table 7.15, the absolute percentage error is below 2% for FF and LUTs, with a latency ratio of 1.025.

	HLS tool					**Model**				
HD	BRAM	DSP	FF	LUT	Latency	BRAM	DSP	FF	LUT	Latency
A.	19	2	497	697	53379	19	2	508	466	52068
B.	19	2	579	852	53557	19	2	1218	789	52670
C.	17	2	13715	23651	126707	17	2	10556	21887	138703

Table 7.14: Metric estimation through HLS tool and MARTE for re-ranking algorithm. The acronyms used in the table are: HD: Hardware design. [A]. No directives. [B]. Pipeline II=12 (loop 1.1) [C]. Pipeline II=12 (loop 1.1) + Pipeline II=6 (loop 2) + Array Partition complete

	P_{error} [%]				AD_L [clk]	L_{ratio}
HD	BRAM	DSP	FF	LUT		
A.	0.000	0.000	0.005	0.197	1,311	1.025
B.	0.000	0.000	0.273	0.054	887	1.017
C.	0.000	0.000	1.349	1.506	11,996	0.914

Table 7.15: P_{error} , AD_L, and L_{ratio} re-ranking algorithm. The acronyms used in the table are: HD: Hardware design. [A]. No directives. [B]. Pipeline II=12 (loop 1.1) [C]. Pipeline II=12 (loop 1.1) + Pipeline II=6 (loop 2) + Array Partition complete

Table 7.16 presents the prediction error for the re-ranking algorithm, considering the estimations reported by MARTE and the results after P & R, showing a P_{error} lower than 3% for BRAM, while the rest of the resources exhibit a P_{error} below 1%.

	BRAM	**DSP**	**FF**	**LUT**
Vivado (P & R)	11	2	754	976
MARTE	19	2	579	852
P_{error} [%]	2.77	0	0.075	0.103

Table 7.16: Re-ranking (implementation of [B] option from Table 7.14). P_{error} considering MARTE estimation and P & R report (Vivado 2021.1.1).

7.4.5 Discussion

The results above showed that MARTE predicted the performance of the designs with a latency ratio nearby. For the basic operations (multiplication, matrix multiplication, and FIR filter), the prediction error was below 1%, estimating with high precision this metric.

Regarding the PSD for cosmic rays classification, the P_{error} was below 2% for the comparison between MARTE and HLS. Considering MARTE and P & R reports, P_{error} was lower than 7% for the LUT resource, which is one of the resources found in large quantities. For DSP and BRAM, P_{error} was below 2%.

For the automatic pest classification based on CNN architecture, the P_{error} obtained for BRAM and LUT was below 50%. Nevertheless, when comparing MARTE and P & R reports, the P_{error} was below 6% for FF and lower than 2% for BRAM.

Finally, for the re-ranking algorithm, P_{error} for LUT and FF was below 2% when comparing with HLS results and lower than 3% for BRAM considering P & R reports.

It is worth mentioning that MARTE does not execute the synthesis or place and route tools in the prediction process. Moreover, MARTE was developed targetting HLS tool without considering P & R reports.

7.5 Assessment of MARTE DSE engine

One of the challenges when developing a design space explorer is the ability for suggesting different combinations of directives to obtain an optimal (or suboptimal) hardware design with a good compromise between area and latency. MARTE, through the DSE engines, is able to propose to the hardware developer a set of configuration, avoiding the execution of HLS in the loop for each

of them.

This section presents the performance evaluation of the MARTE DSE engines. In the first place, the assessment of the DSE based on a single-objective BO for latency optimization is presented. Subsequently, the results obtained through the DSE based on evolutionary multi-objective optimization are exposed. To this end, multiplication, matrix multiplication, FIR filter, MLP, and re-ranking applications are considered.

7.5.1 Assessment of the DSE engine based on BO

BO configuration: surrogate function: GP, acquisition function: EI, eliminate duplicates: enable, termination criteria: 300 (maximum number of iterations).

The analysis of the DSE based on BO implemented as a single-objective optimization problem is presented, considering the directive combinations proposed by the explorer. The objective function is the latency estimation, defined by the corresponding analytical model incorporated in MARTE.

The objective space is presented in Fig. 7.1, considering multiplication, matrix multiplication, and FIR filter applications. The red mark represents the values obtained through the HLS tool, corresponding to the ones presented in Section 7.4.1, and the blue dots are the solutions retrieved by the EMO DSE engine. For multiplication and matrix multiplication, the solutions reported with HLS tools match with solutions in the objective space, with an error lower than 1% for the area and with the values of latency near the HLS solutions. In the case of FIR filter, it can be remarked that the solutions with higher values

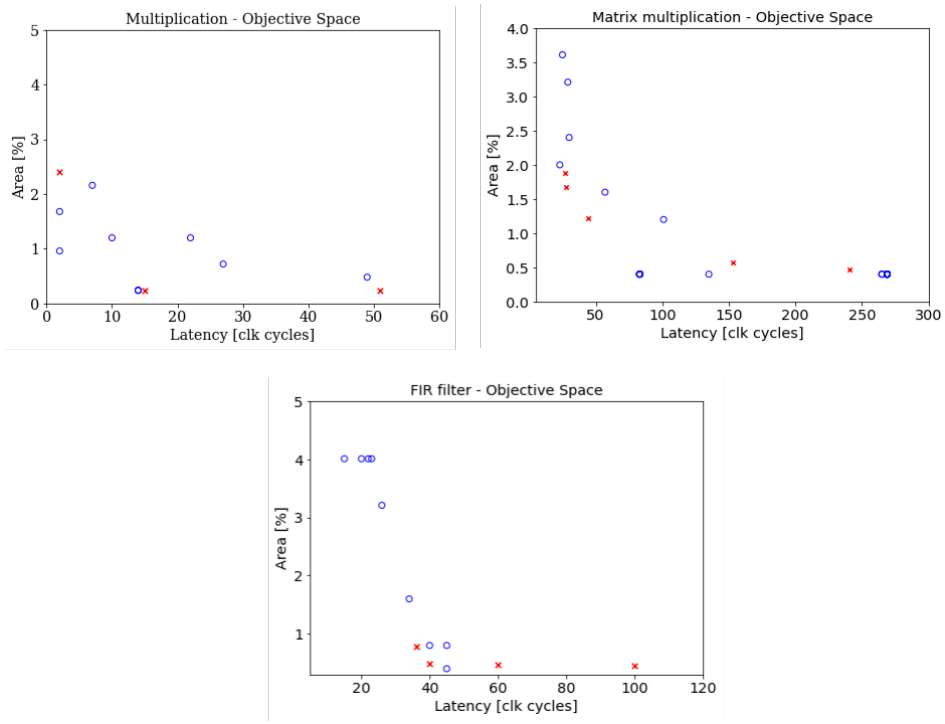


Figure 7.1: DSE based on BO. Objective space for multiplication, matrix multiplication, and FIR filter.

Table 7.17 presents the configuration suggested by the DSE engine for each application.

Algorithm	**Best configuration - Latency optimization**
Mult	D = 2, uFactor = 10, AP = complete
Matrix mult	AP = complete; (loop1:row) D = 2 uFactor = 2
FIR	AP = block with factor 3; D = 1 II=3
MLP	AP = complete , D=1 II=1 for each external loop (Product1, ResetAccum, Accum1, Result)
QS_100T_32FP	AP = complete (structures with less than 1000 elements) D=1 II=13 (MC loop:product), D=1 II=6 (loop:SC)

Table 7.17: Configurations provided by MARTE DSE based on single-objective BO for latency optimization. The acronyms are: D=0 No directive, D=1 Pipeline, D=2 Unroll, AP: array partition.

7.5.2 Assessment of the DSE engine based on EMO

NSGA-II configuration: Number of generations: 300, population size: 60, number of offspring: 5, sampling: integer random, cross-over: simulated binary crossover (SBX) [260], mutation: polynomial mutation (PM) [260], eliminate duplicates: enable, termination criteria: number of generations, seed: random initialization.

This section presents the analysis of the DSE based on EA implemented as a multi-objective optimization problem. The objective functions are latency and area estimation, defined by the corresponding analytical models. First, the convergence of the DSE is presented, followed by the recommended directives for each application. Then, runtime and compatibility assessment are exposed.

Convergence

The convergence of EMO DSE for the basic applications is presented in Fig. 7.2, obtained through the running metric. The convergence is plotted every five generations (blue, orange, and green lines corresponds to generations 5, 10, and 15 respectively). The algorithm significantly improved for multiplication algorithm from the 2nd to the 3rd generation. Regarding matrix multiplication, the progress to convergence is noticeable from the 8th to the 9th generation. In the case of the FIR filter, the algorithm showed the first improvement from the 3rd to the 5th generation. Multiplication and FIR filter applications took lesser generations to reach a stable value than matrix multiplication.

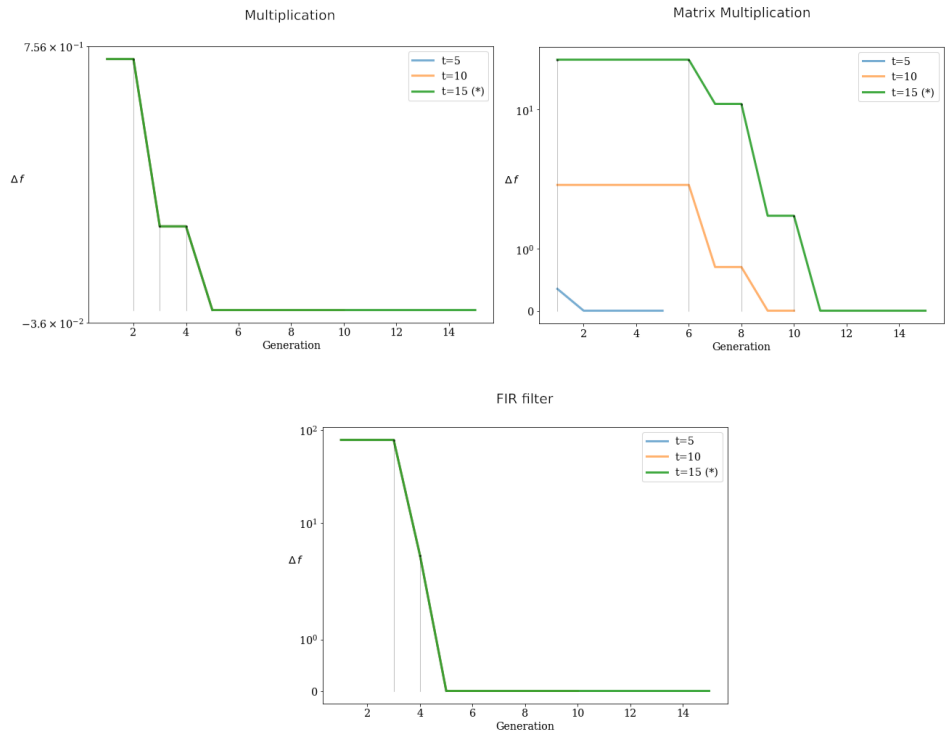


Figure 7.2: Running metric. Convergence EMO DSE for basic applications.

Fig. 7.3 presents the convergence curves for the highly demanding applications. The PSD based on MLP algorithm shows an increase in the convergence between generations until the 9th. After this, the curve exhibits a fast improvement until the 10th generation, leading to the algorithm's convergence. For the re-ranking application, the first improvement was initiated from the 3rd to the 4th generation. Then, the progress to convergence continued until the 12th generation. Then, the improvement tends to be constant. The algorithms converged fast in the first four generations in the basic applications. Regarding the highly demanding applications, convergence was reached after nine generations.

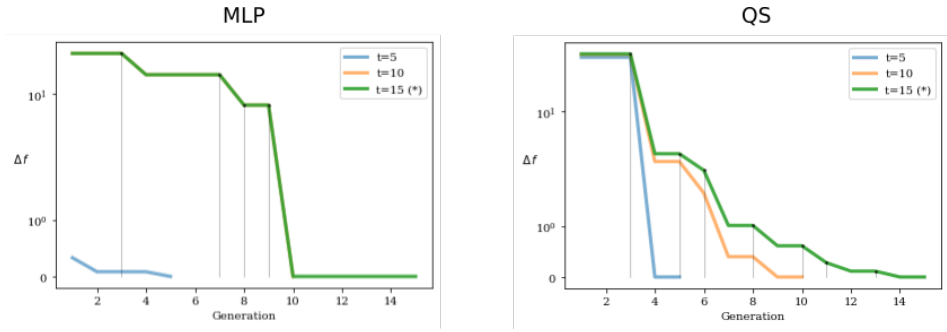


Figure 7.3: Running metric. Convergence EMO DSE for highly demanding applications.

High-performance configuration

A desirable functionality for a DSE engine is the suggestion of a suitable combination of directives with a good trade-off between latency and area. Table 7.18 presents the configuration suggested by MARTE for each application when de DSE engine is based on EMO.

Algorithm	**Best configuration - Latency-area optimization**
Mult	D = 2, uFactor = 10, AP = complete
Matrix mult	(loop3) D = 2 , uFactor = 2 , AP = complete
FIR	D = 1, AP = 3, uFactor = 1
MLP	AP = complete , D = 1 II = 1 for each external loop (Product1, ResetAccum, Accum1, Result) D = 2 for each internal loop (Product2, Acumm2)
QS	(MC loop:outer_loop) D=1, II=20; (SC loop) D=1, II=8

Table 7.18: MARTE EMO DSE efficient configurations. The acronyms are: D=0 No directive, D=1 Pipeline, D=2 Unroll, AP: array partition.

The objective spaces are presented in Fig. 7.4, considering multiplication, matrix multiplication, and FIR filter applications. The red mark represents the values obtained through HLS tool, presented in Section 7.4.1, and the blue dots represent the solutions retrieved by the EMO DSE engine. As can be noticed, the difference between blue and red marks is lower than 0.25% in the case of the area and lesser than two clock cycles in regard to the latency.

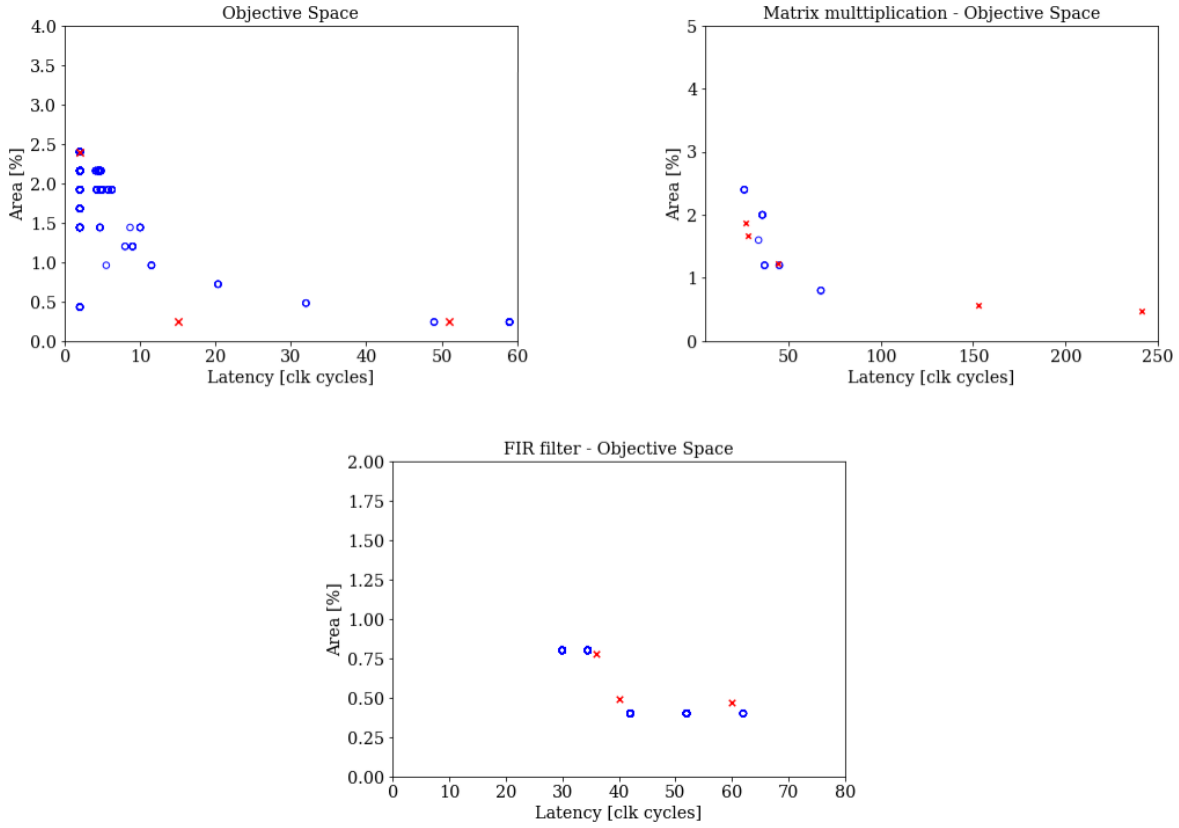


Figure 7.4: Objective space EMO DSE for basic applications.

Fig. 7.5 shows the objective space for the highly demanding applications. As the optimization algorithm searches for the optimal solution, the final objective space may include some possible (non-optimal) solutions. Fig. 7.5 presents The objective space retrieved for the re-ranking algorithm. The red marks, placed in a value for area around 20%, represent the solutions obtained manually by tuning the directives. These are not optimal because they imply an increment of area and latency. Hence, the DSE engine avoids retrieving them. The same behavior appears for matrix multiplication in the objective space from Fig. 7.4.

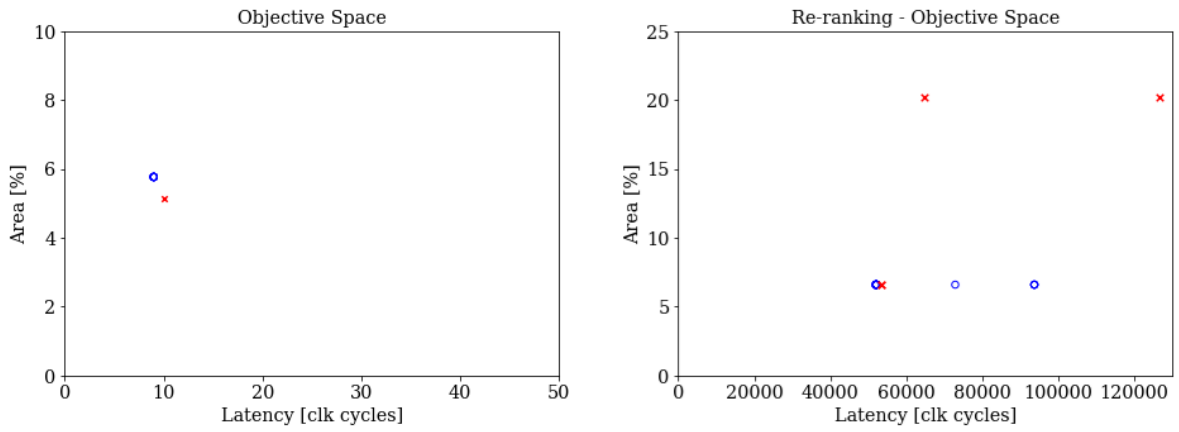


Figure 7.5: Objective space EMO DSE for MLP and pre-ranking algorithms.

From the objective spaces presented in Fig. 7.4, 7.5, it can be noted that MARTE obtains with high precision the solutions that have the lowest latency of the explored space.

The variability of the retrieved objective space can be associated with the genetic operations (mutation, crossover, selection), that may need to be adjusted for each application if the complete set of optimal solutions need to be obtained [28]. Given that this research aims to obtain a set of solutions that serve as a starting point to avoid manual exploration of the search space, the retrieved solutions present a good compromise between latency and area.

7.6 MARTE runtime analysis

An essential aspect of DSE is the time to explore the whole design space (or a section of it), retrieving a set of feasible solutions. Table 7.19 presents the runtime obtained with the different tools: single-objective BO, EMO, EMO without termination, MARTE single estimation, and HLS single execution (directives applied). Even if the DSE based on EA and BO proposed reasonable solutions, BO is slower than both EA implementations (with and without termination criterion). In this experiment, a section of the search space was sampled considering that DSE engines comprises 2 objective functions and 3 different variables per loop linked with the directives (3 type of directive, II value, and UF).

Considering the MLP application, HLS took 58.58 seconds to obtain the resource estimation with directives applied to obtain the highest parallelization level (unroll, pipelining at top-level

function, and array partition). In contrast, for the same scenario, MARTE took 0.033 seconds to perform the performance estimation. Regarding the DSE engine, to find a set of feasible solutions, MARTE DSE employed 235 seconds with termination criterion, representing a smaller runtime compared to HLS execution to obtain one single estimation.

Application	**Single-objective BO**	**EMO**	**EMO nT**	MARTE **single estimation**	**HLS single execution**
Mult	434	83.18	9.31	0.015	7.91
Matrix mult	207	332	83.21	0.004	8.56
FIR	358	71.4	8.52	0.025	4.87
MLP	187.68	235	35.6	0.033	58.58
QS_100T_32FP	479	180.7	22.4	0.02	8.51

Table 7.19: Runtime measured in seconds. HLS single execution with directives. Single-objective BO with stopping-criterion 100 iterations.

7.7 MARTE compatibility analysis

One of the main challenges for a performance estimator is that it should guarantee compatibility among different HLS tool versions. For this assessment, the target device was xczu9eg-ffvb1156-2-e, considering a target clock of 5ns. The available resources for the device are presented in Table 7.20. In addition, the hardware developer should check the inherent rules related to the use of directives and the HLS tool version.

ZCU102 - Available resources			
BRAM	DSP	FF	LUT
1824	2520	548160	274080

Table 7.20: Available resources on ZCU102 development board.

Table 7.21 compares MARTE and the results obtained with Vivado HLS 2019.1.1, using only the basic cases (multiplication, matrix multiplication, and FIR filter) to demonstrate its portability among different versions, and the corresponding P_{error} and L_{ratio} are shown in Table 7.22. The

latter shows that the P_{error} is below 0.5%, while the L_{ratio} is lesser than 1.5. The use of micro-benchmarks helps to grant compatibility among different HLS tool versions.

	HLS tool					**Model**				
Multiplication										
HD	BRAM	DSP	FF	LUT	Latency	BRAM	DSP	FF	LUT	Latency
A.	0	3	268	222	81	0	3	205	209	79
B.	0	30	1515	1483	4	0	30	1345	1411	4
C.	0	3	395	282	18	0	3	251	213	13
Matrix multiplication										
HD	BRAM	DSP	FF	LUT	Latency	BRAM	DSP	FF	LUT	Latency
D.	0	5	616	629	430	0	5	542	590	432
E.	0	5	1320	912	57	0	5	1300	801	59
F.	0	30	3675	3081	32	0	30	2760	2956	31
FIR filter										
HD	BRAM	DSP	FF	LUT	Latency	BRAM	DSP	FF	LUT	Latency
G.	0	5	699	583	177	0	5	528	522	179
H.	0	3	922	1070	63	0	5	1034	814	63
I.	0	3	662	760	63	0	5	610	532	64

Table 7.21: Comparison results between MARTE and Vivado HLS 2019.1.1, targetting xczu9eg-ffvb1156-2-e part. The acronyms used in the table are: HD: Hardware design. **Multiplication:** [A]. No directives, [B] AP + Unroll. [C] Pipeline II=1, **Matrix Multiplication:** [D]. No directives. [E]. Pipeline II=3 (Loop: Col) + AP. [F]. Pipeline II=3 (Loop: Row) + AP. **FIR filter:** [G]. No directives. [H]. Pipeline II = 3 + AP complete. [I]. Pipeline II = 3.

Error						
Multiplication						
	P_{error} [%]				AD_L [clk]	L_{ratio}
HD	BRAM	DSP	FF	LUT		
A.	0	0	0.011	0.005	2	1.03
B.	0	0	0.031	0.026	0	1.00
C.	0	0	0.026	0.025	5	1.38
Matrix multiplication						
D.	0	0	0.013	0.014	2	0.99
E.	0	0	0.004	0.040	2	0.97
F.	0	0	0.167	0.046	1	1.03
FIR filter						
G.	0	0	0.031	0.022	2	0.98
H.	0	0.079	0.020	0.093	0	1.00
I.	0	0.079	0.010	0.083	1	0.98

Table 7.22: P_{error}, AD_L, and L_{ratio} for the basic applications considering MARTE and Vivado HLS 2019.1.1. The acronyms used in the table are: HD: Hardware design. **Multiplication:** [A]. No directives, [B] AP + Unroll. [C] Pipeline II=1. **Matrix Multiplication:** [D]. No directives. [E]. Pipeline II=3 (Loop: Col) + AP. [F]. Pipeline II=3 (Loop: Row) + AP. **FIR filter:** [G]. No directives. [H]. Pipeline II = 3 + AP complete. [I]. Pipeline II = 3.

7.8 Summary of the chapter

This chapter presented MARTE performance assessment, considering basic and highly demanding applications for evaluating the cost models, DSE engines, runtime, and compatibility.

The results regarding analytical models for resource and latency estimation, compared with HLS tool, exhibited a prediction error lesser than 1% for resource utilization and a latency ratio below 2 for the basic applications. At the same time, the evaluation of MARTE for PSD, CNN, and QS reported a prediction error lesser than 2% for area estimation and a latency ratio below 1 for PSD and QS. Nevertheless, for CNN algorithm, the prediction error was near 49%. However, when compared MARTE with P & R reports for the highly demanding applications, the prediction error was below 6.3% for LUT and FF, which are the most abundant resources.

Furthermore, MARTE estimated the latency for all the applications with high precision because the ratio obtained was near 1, compared with HLS tool.

DSE engines based on single- and multi-objective optimization were evaluated when providing hardware designs with a good compromise between area and latency. One of the differences between both engines was the runtime to obtain a set of feasible solutions, being DSE-based on EMO the fastest. This behavior was possible because of the use of MARTE cost models in the design exploration loop instead of the HLS tool to synthesize each design point. Nevertheless, one of the essential things to reaching the balance between execution time and exploration-exploitation of the search space is the high accuracy that should have the cost models.

Since one of the main drawbacks of evolutionary algorithms is that the search can stop at a local minimum, the single-objective optimization based on BO acts as a redundant algorithm to explore the search space, which increases the runtime. Furthermore, the different genetic operators allow a better convergence, trying to prevent the algorithm from remaining at a local minimum.

Given that this research aims to obtain a set of solutions that serve as a starting point to avoid the manual exploration of the search space, the retrieved solutions by MARTE present a good compromise between latency and area, considering both DSE sub-engines. Furthermore, the results of compatibility and runtime show the benefits of using the design flow to improve productivity when designing efficient hardware through HLS tools.

The next chapter will present the integration of MARTE with the Roofline model.

Chapter 8

Integration of MARTE with the state of the art

This chapter presents the integration of MARTE with Roofline, the leading parallel computing model adapted to FPGA architectures, showing the benefits of performing this combination. The target application is the pulse shape discriminator based on MLP architecture.

8.1 Roofline Model

Roofline is a parallel computing model adapted for FPGA architectures, as introduced in Section 3.2.5. This model can be considered complementary to MARTE because it provides information regarding operational intensity and attainable performance, avoiding the direct estimation of latency and resource utilization. Based on this assumption, this chapters present the integration between MARTE and Roofline. MARTE is employed to estimate the latency and the resource utilization of a PE. When working with neural networks, Roofline model can provide information about the size of the network, the trade-off between FPGA hardware resources linked to the attainable performance, and the impact on the overall system due to quantization.

The peak performance PC is re-defined in terms of the total number of operations n_{op} and the desired frequency f for the programmable logic, linked to the ideal implementation, as it is shown

Section 8 is the result of an intership relized on October 2022 at NECST Laboratory, Dipartimento di Elettronica, Informazione e Bioingengeria (DEIB), Politecnico di Milano.

in the Eq. 8.1.

$$PC = \sum n_{op} \times f \tag{8.1}$$

The horizontal roof depends on the computational performance, and with SoC-FPGA architectures, it is a dynamic roof tied to the algorithm and resource utilization. For a given kernel, the performance cannot be higher than that of the horizontal roof because that is the hardware limit.

The computational intensity (CI) links the algorithm to the architecture. It is computed as the ratio between the sum of the arithmetic, indexing, and comparison operations and the memory traffic (read and write operations). The last parameter is associated with the number of bytes that should be read or written during the computations. Hence, the number of operations is linked to the resources available in the FPGA.

The Roofline model is constructed using the information provided by MARTE when the application is based on DNN and implemented using hls4ml. The tree data structure is the input of MARTE, which is used to estimate the latency and resource utilization. Hence, the roof of the Roofline model is tied to the hardware required to implement the DNN on the FPGA.

Furthermore, the weights and biases of the DNN models are stored in the on-chip memory, reducing the bandwidth pressure and shifting the bottleneck to the available computational resources inside the chip.

With hls4ml, the most affected resource for computation is the number of DSPs. Nevertheless, the resource utilization is modified when the reuse factor increases: DSPs are exchanged with LUTs. Therefore, two plots based on roofline can be obtained: one linked to DSP utilization and the other to LUTs, since both resources affect the position of the horizontal roof, tied with the attainable performance. The hardware resource constraints the optimal performance with the highest ratio, affecting the replication factor for a given PE.

8.1.1 Pulse shape discriminator

The Roofline model is constructed to analyze the attainable performance for each distilled architecture, as presented in Table 6.1 of Section 6.2.

The slope of the plot gives information about the memory bandwidth, correlated with the device's throughput. For an UltraScale+ ZCU102 development board, the theoretical DDR through-

put is 2400 MT/s × 64-bit, and this theoretical performance for ZCU102 is larger than the available throughput of a single AXI port. In practice, 75% of the theoretical value is considered. The aggregated throughput between the PS and PL is 12.8 GB/s (four AXI ports at 2x1.6 GB/s each) [261]. For the PSD implementation presented in this Chapter, the throughput corresponds to one of the high-performance (HP0) ports.

The Roofline model is adapted with the aim of assisting the hardware developer in selecting the distilled network to be deployed on the SoC-based FPGA, selecting the ideal implementation in hardware as the starting point: maximum precision and no data dependencies. Therefore, the ideal upper bound for attainable performance is established [127]. As a starting point, Fig. 8.1 presents the attainable performance of each student's architecture from Table 6.1 considering 32-bit floating-point.

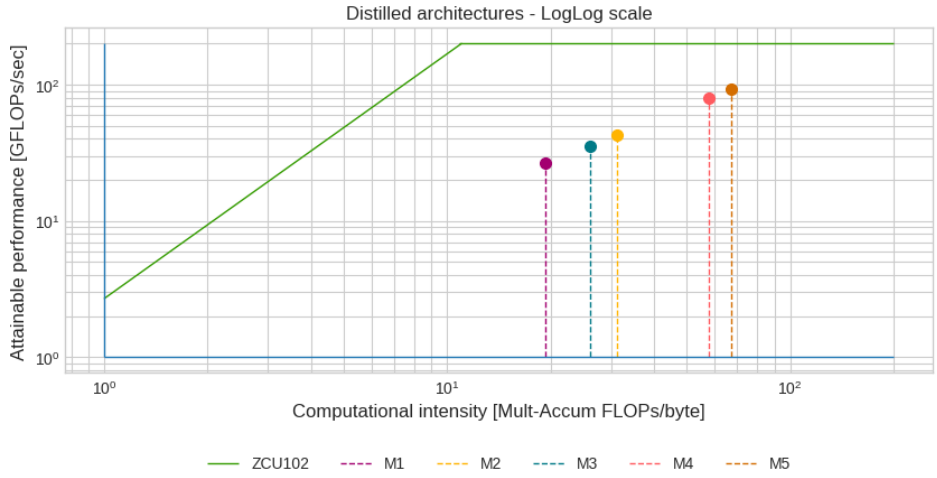


Figure 8.1: Roofline model for MLP-based models targeting ZCU102 platform.

For an FPGA/SoC implementation with data structures defined in 32-bit floating point, the peak performance is affected mainly by DSP and multiply-accumulate operations. Based on this, the hardware developer can choose to implement the one with the highest performance to exploit the architecture or one with lower performance if design constraints limit the application.

Once the student network is selected, and due to that hls4ml supports fixed-point operations and data structures, the Roofline is generated for 32-bit fixed-point precision.

Furthermore, for 32-bit fixed-point precision, the latency and resource utilization associated with each elementary arithmetic operation are presented in Table 8.1, considering multiplication

and addition, which are the most used operations in an MLP architecture.

Operation	BRAM	DSP	LUT	FF	Clock cycles
Multiplication	-	3	20	0	1
Addition	-	0	39	0	1

Table 8.1: Resource and latency estimation for the elementary operations obtained through HLS tool, considering 32-bits fixed point precision.

The Roofline model has the y-axis defined in GFLOPs/sec, which directly correlates with the resource utilization of the FPGA/SoC device. From this premise, the y-axis can be adapted to fixed-point operations per second, which means that the y-axis is redefined as GFPOPs/sec.

The peak performance defines the roof of the new Roofline, linked to the hardware implementation and the platform. Fig. 8.2 presents the attainable performance of the M1 model defined with 32-bit fixed-point precision. As can be noticed, the M1's roof is distant from the peak performance because, in this case, it is tied to the multiply-accumulate operations and their impact on DSP utilization.

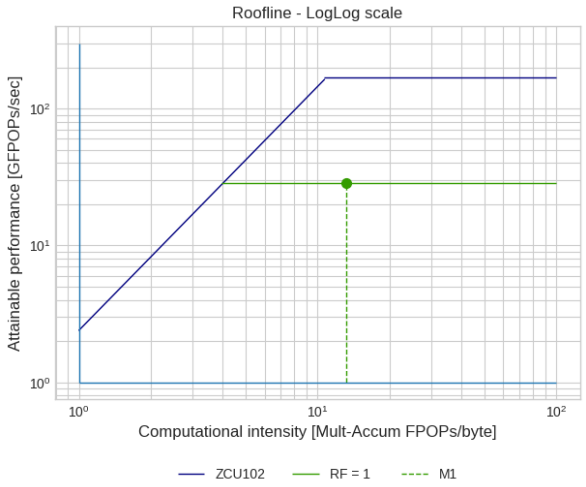


Figure 8.2: Roofline model for MLP M1 model 32-bit fixed-point precision, targeting ZCU102 platform.

Fig. 8.3 presents the impact of the RF for the 32-bit fixed-point implementation. By varying the

RF in the hls4ml package, the attainable performance of the model will decrease. This means that, for the same computational intensity, fewer DSPs will be needed, allowing the implementation of the application in low-end devices without losing accuracy but compromising performance in terms of latency.

The position of the horizontal roof is linked to the RF and the DSPs, impacting the resource utilization for a given SoC-based FPGA. The highest performance is achieved for a ruse factor of 1, which corresponds to the plot presented in Fig. 8.2, while the lower position of the roof corresponds to a reuse factor of 32.

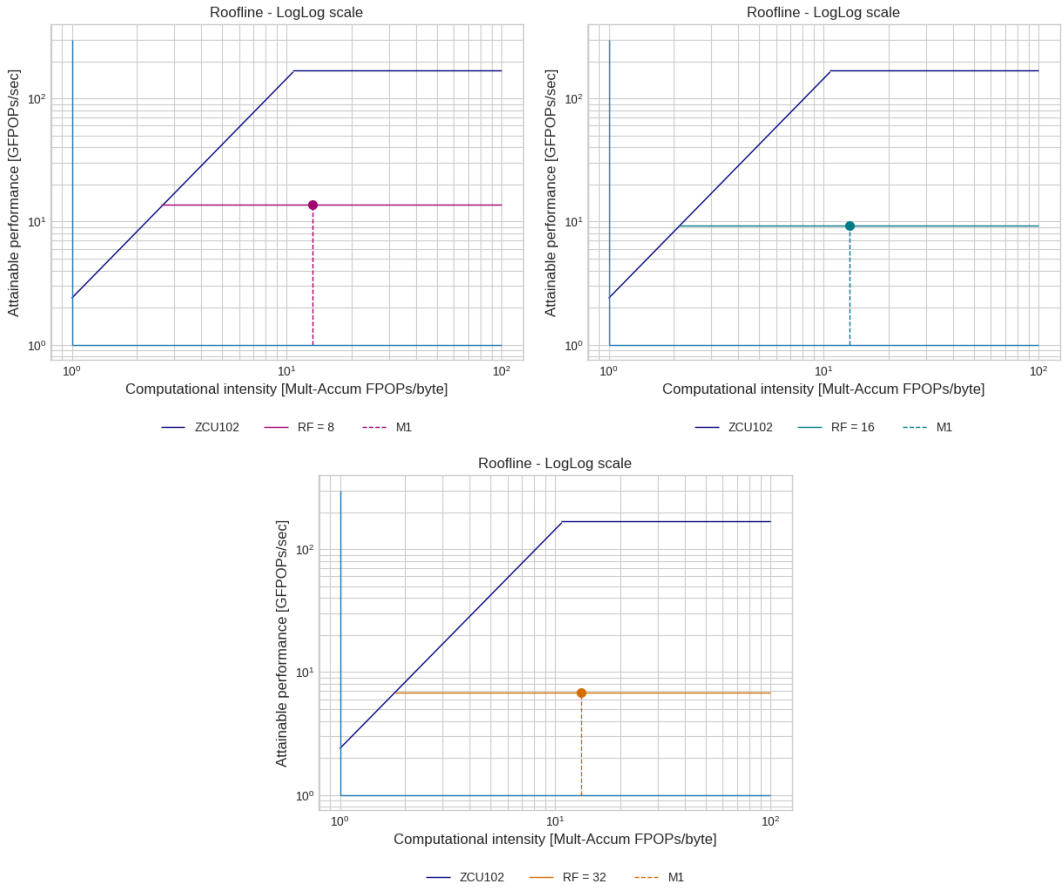


Figure 8.3: Roofline model for MLP M1 model 32-bit fixed-point precision, targeting ZCU102 platform. Reuse factor impact in the attainable performance.

Fig. 8.4 presents the relationship between the reuse factor and the latency for ZCU102 and Zedboard devices, showing that increasing the reuse factor implies an increase in latency.

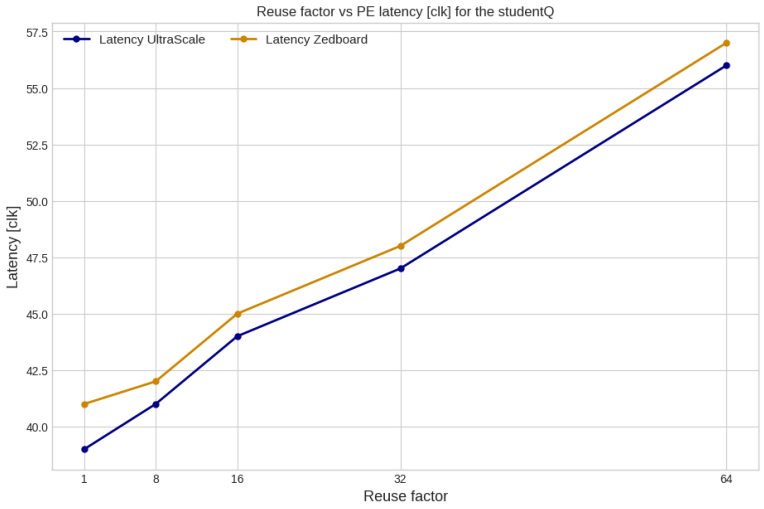


Figure 8.4: Reuse factor vs latency for MLP architecture.

Moreover, the scalability of the system is affected by RF, which means that the PE can be replicated in the PL part at expenses of resource utilization and, in consequence, latency. This can be observed in Fig. 8.5 for ZCU102 and Zedboard devices. The difference in resource utilization between both architectures is evident, while it is possible to replicate the PE by a factor of 35 for ZCU102, in the case of the Zedboard is a factor of 6. This exposes the effect of the RF showing that the same architecture can be deployed in high- and low-end FPGA architectures.

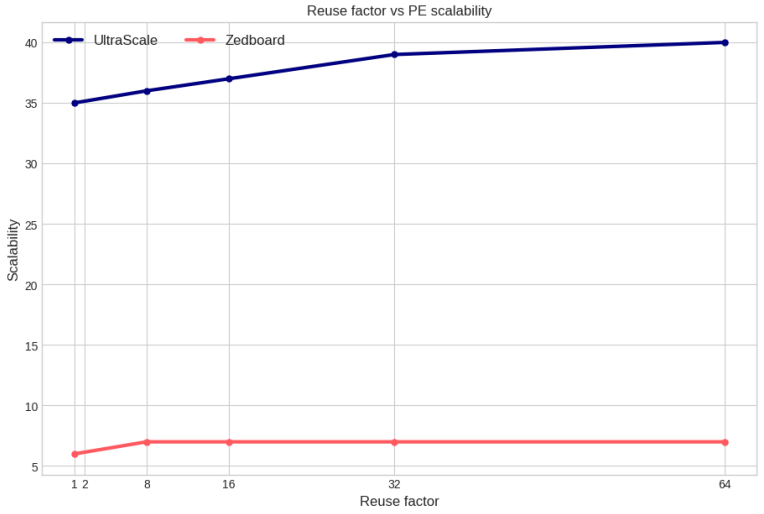


Figure 8.5: Reuse factor vs scalability for MLP architecture.

8.2 Discussion

As presented in Chapter 4.1, contributions in the literature such as [125,127] showed the benefits of the Roofline model applied to FPGA/SoC devices, to estimate the attainable performance. According to Section 3.3, a parallel computing model can be coupled with a performance estimator for FPGA/SoC, where latency and area are the main metrics to optimize. Thus, the estimator provides the information needed to build the parallel model, specially when the target architecture is based on reconfigurable logic.

In the particular case of Roofline, MARTE is executed for modeling the FPGA logic, estimating the resource utilization and the latency for a given application, avoiding the execution of the synthesis process in the estimation flow. Thus, the combination of both estimators allows obtaining the attainable performance of the system, improving the productivity of the developers.

Moreover, MARTE includes parts of COMBA [85] in the cost models for latency and area, exposing the potentiality when combining existing estimators in the literature, improving the productivity cycles of the hardware designer.

8.3 Summary of the chapter

This chapter presented the integration between MARTE and Roofline model, the leading parallel computing model adapted to FPGA architectures. This combination allows observing the attainable performance and operational intensity considering latency and resource estimation obtained through MARTE.

In addition, when considering DNN implementations, information about the ML model, the impact of quantization, and the trade-off between FPGA resources (mainly DSP and LUT) can be inferred from MARTE and Roofline.

Based on the presented results, this combination can be improved by ...

Chapter 9

Conclusions

This thesis presented the use of SoC-based FPGA architectures for image analysis and other highly demanding applications focused on obtaining efficient hardware designs through HLS tools. Thus, two complementary strategies were considered: using **ad-hoc techniques** and **performance estimators**.

Regarding **ad-hoc techniques**, three highly demanding applications were accelerated through HLS tools: pulse shape discriminator for cosmic rays, automatic pest classification, and re-ranking for information retrieval. It was observed how ad-hoc techniques, such as compression and machine learning, can lead to optimal hardware implementation with a good compromise between efficiency, effectiveness, memory footprint, and inference time. For all the cases of study, a fully-on-chip deployment was achieved through HLS tools, considering code restructuring and directive selection to obtain a hardware design with a good trade-off between latency and resource utilization.

Furthermore, the use of a methodology for DNN compression based on hyperparameter tuning, quantization, pruning, and knowledge distillation allowed the implementation of efficient classifiers in applications based on neural networks.

Regarding each application, it can be said that:

- The pulse shape discriminator was implemented through an MPL architecture, aimed to perform the signal classification in an electronic front-end for particle detection, obtaining an inference time of 14 clock cycles for ZCU102 board @ 200 MHz.

- A CNN-based classifier for precision agriculture for automatic pest classification was intro-

duced, showing the potential of transfer learning when working with a dataset directly related to the application, with a compression ratio of 7409x, in the number of parameters, for the student architecture. In this application, the on-chip deployment stressed the LUT resource with 21% of utilization, which means a reduced memory footprint on an embedded system.

- A re-ranking algorithm for information retrieval was implemented, taking into account 32-bit floating-point and 8-bit fixed-point data structures. Moreover, this study case exhibited the benefits of code restructuring techniques when targeting SoC-based FPGA.

The different applications exhibited the benefits of compression techniques when targetting SoC-based FPGA to improve memory footprint and inference time.

Regarding the second strategy focused on **performance estimators**, to bridge the gap between the application and FPGA/SoC architecture, MARTE was proposed as a compressive performance estimator for reconfigurable hardware accelerators. MARTE is composed of a cost model and a DSE engine. The former aims to predict the objective functions of latency (clock cycles) and area (defined by LUT, BRAM, DSP, and FF), considering directives for loop handling and array partitioning, typically employed to improve latency, performance, throughput, and area. The DSE engine is based on two sub-engines to explore the design space: one based on single-objective BO and one based on EMO, retrieving a set of feasible solutions for the user. A set of rules defined in the HLS tool guided the search space exploration.

Since one of the main drawbacks of evolutionary algorithms is that the search can stop at a local minimum, the single-objective optimization based on BO acts as a redundant algorithm to explore the search space at the expense of an increase in the runtime. Furthermore, the different genetic operators allow a better convergence, trying to prevent the algorithm from remaining at a local minimum. Nevertheless, DSE based on EMO can converge quickly to solutions with a good compromise between latency and area. This research was focused on obtaining performance estimations at the macro level and fast convergence to obtain a set of feasible solutions that can assist the developer in increasing productivity in the early stages of the designs. From this premise and based on the obtained results, it can be affirmed that DSE based on EMO is a suitable approach.

The results showed that MARTE predicted the designs' performance with a latency ratio nearby one. For the basic operations (multiplication, matrix multiplication, and FIR filter), the P_{error} was

below 1%.

Regarding the highly demanding applications, for the PSD, the P_{error} was below 2% for the comparison between MARTE and HLS. Considering MARTE and P & R reports, P_{error} was lower than 7% for LUT resource, which is one of the resources found in large quantities; for DSP and BRAM, P_{error} was below 2%. For the automatic pest classification based on CNN architecture, the P_{error} obtained for BRAM and LUT was below 50%. Nevertheless, when comparing MARTE and P & R reports, the P_{error} was below 6% for FF and lower than 2% for BRAM. Finally, for the re-ranking algorithm, P_{error} for LUT and FF was below 2% when comparing with HLS results and lower than 3% for BRAM considering P & R reports.

Furthermore, one way of granting compatibility between MARTE and different versions of HLS tools is through micro-benchmarks and the adequate incorporation of the HLS rules.

In this research, obtaining a set of solutions that serve as a starting point is desirable to avoid the manual exploration of the search space. The retrieved solutions by MARTE present a good compromise between latency and area, considering both DSE sub-engines. Furthermore, the results of compatibility and runtime show the benefits of using the design flow to improve productivity when designing efficient hardware through HLS tools.

In addition, it was presented the feasibility of the integration of MARTE with Roofline model, the leading parallel computing model adapted to FPGA architectures.

Based on the presented research, it can be said that it is possible to develop a methodology for SoC-based FPGA architectures composed of analytical models and integrated with a DSE engine based on mathematical programming, guiding the exploration process through HLS rules to estimate performance metrics, while improving the productivity of the hardware developers.

9.1 Future directions

As future research can be considered:

- Integration of MARTE in strategies to perform compression of ML-based models, being able to adapt the ML models to the hardware in the training step.
- Extend the compression methodology to other types of ML architectures.
- Integration of MARTE with other parallel computing models, such as BSP.

- Exploration of others solvers and methods for DSE based on EMO.
- Inclusion of more directives and modes to MARTE, to have a robust estimator.
- Generation of a library that includes MARTE so that it can be accessed by the developers and easily integrated into the design flow.
- Automation of the source code translation into the tree data structure.
- Verification of MARTE's functionality with other cases of study.
- Creation of a custom mechanism to balance the exploitation-exploration phase in single- and multi-objective optimization.

Bibliography

[1] R. S. Molina, V. Gil-Costa, M. L. Crespo, and G. Ramponi, "High-level synthesis hardware design for fpga-based accelerators: Models, methodologies, and frameworks," *IEEE Access*, vol. 10, pp. 90 429–90 455, 2022.

[2] Z. Wan, B. Yu, T. Y. Li, J. Tang, Y. Zhu, Y. Wang, A. Raychowdhury, and S. Liu, "A survey of FPGA-based robotic computing," *CoRR*, vol. abs/2009.06034, 2020.

[3] M. Bakiri, C. Guyeux, J.-F. Couchot, and A. K. Oudjida, "Survey on hardware implementation of random number generators on FPGA : Theory and experimental analyses," *Computer Science Review*, vol. 27, pp. 135 – 153, 2018.

[4] A. Ebrahimi and M. Zandsalimy, "Evaluation of FPGA hardware as a new approach for accelerating the numerical solution of CFD problems," *IEEE Access*, vol. 5, pp. 9717 – 9727, 2017.

[5] K. Guo, S. Zeng, J. Yu, Y. Wang, and H. Yang, "[DL] a survey of FPGA-based neural network inference accelerators," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 12, no. 1, 2019.

[6] R. Nane, V. M. Sima, C. Pilato, J. Choi, B. Fort, A. Canis, Y. T. Chen, H. Hsiao, S. D. Brown, F. Ferrandi, J. H. Anderson, and K. Bertels, "A survey and evaluation of FPGA high-level synthesis tools," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 10, pp. 1591–1604, 2016.

[7] X. Inc., "Vivado design suite user guide: High-level synthesis. UG-902," 2020.

[8] J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K. A. Vissers, and Z. Zhang, "High-level synthesis for FPGAs: From prototyping to deployment," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 4, pp. 473–491, 2011.

[9] Intel Corp., "Intel high level synthesis compiler, best practices guide. UG-20107," 2020.

[10] A. Canis, J. Choi, M. Aldham, V. Zhang, A. Kammoona, J. H. Anderson, S. D. Brown, and T. S. Czajkowski, "LegUp: high-level synthesis for FPGA-based processor/accelerator systems," in *International Symposium on Field Programmable Gate Arrays, FPGA*, 2011, pp. 33–36.

[11] C. Pilato and F. Ferrandi, "Bambu: A modular framework for the high level synthesis of memory-intensive applications," in *International Conference on Field programmable Logic and Applications, FPL*, 2013, pp. 1–4.

[12] R. Nane, V. Sima, C. Pilato, J. Choi, B. Fort, A. Canis, Y. T. Chen, H. Hsiao, S. Brown, F. Ferrandi, J. Anderson, and K. Bertels, "A survey and evaluation of FPGA high-level synthesis tools," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 10, pp. 1591–1604, 2016.

[13] Y. Liang, K. Rupnow, Y. Li, D. Min, M. Do, and D. Chen, "High-level synthesis: Productivity, performance, and software constraints," *Electrical and Computer Engineering*, vol. 2012, pp. 649 057:1–649 057:14, 2012.

[14] Khronos OpenCL Working Group, *The OpenCL Specification, Version 1.1*, 2011.

[15] C. Kessler and J. Keller, "Models for parallel computing: Review and perspectives," *Mitteilungen - Gesellschaft für Informatik e.V., Parallel-Algorithmen und Rechnerstrukturen*, vol. 24, pp. 13–29, 2007.

[16] G. Litjens, T. Kooi, B. E. Bejnordi, A. A. A. Setio, F. Ciompi, M. Ghafoorian, J. A. van der Laak, B. van Ginneken, and C. I. Sánchez, "A survey on deep learning in medical image analysis," *Medical Image Analysis*, vol. 42, pp. 60–88, 2017.

[17] S. Li, W. Song, L. Fang, Y. Chen, P. Ghamisi, and J. A. Benediktsson, "Deep learning for hyperspectral image classification: An overview," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 57, no. 9, pp. 6690–6709, 2019.

[18] H. Jang and J.-S. Lee, "Analysis of deep features for image aesthetic assessment," *IEEE Access*, vol. 9, pp. 29 850–29 861, 2021.

[19] M. Feickert and B. Nachman, "A living review of machine learning for particle physics," *CoRR*, vol. abs/2102.02770, 2021.

[20] A. M. Deiana, N. Tran, J. Agar, M. Blott, G. D. Guglielmo, J. Duarte, P. C. Harris, and S. H. et al., "Applications and techniques for fast machine learning in science," *CoRR*, vol. abs/2110.13041, 2021.

[21] S. L. Brunton, B. R. Noack, and P. Koumoutsakos, "Machine learning for fluid mechanics," *Annual Review of Fluid Mechanics*, vol. 52, no. 1, pp. 477–508, 2020.

[22] S. I. Venieris, A. Kouris, and C. Bouganis, "Toolflows for mapping convolutional neural networks on FPGAs: A survey and future directions," *ACM Computing Surveys*, vol. 51, no. 3, pp. 56:1–56:39, 2018.

[23] A. Reuther, P. Michaleas, M. Jones, V. Gadepally, S. Samsi, and J. Kepner, "Survey and benchmarking of machine learning accelerators," in *2019 IEEE high performance extreme computing conference (HPEC)*. IEEE, 2019, pp. 1–9.

[24] A. Reuther, P. Michaleas, M. Jones, V. Gadepally, S. Samsi, and J. Kepner, "Survey of machine learning accelerators," in *2020 IEEE high performance extreme computing conference (HPEC)*. IEEE, 2020, pp. 1–12.

[25] I. Sidelnik, H. Asorey, L. Collaboration *et al.*, "Lago: the latin american giant observatory," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 876, pp. 173–175, 2017.

[26] C. Lucchese, F. M. Nardini, S. Orlando, R. Perego, N. Tonellotto, and R. Venturini, "QuickScorer: A fast algorithm to rank documents with additive ensembles of regression trees," in *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2015, pp. 73–82.

[27] D. Dato, C. Lucchese, F. M. Nardini, S. Orlando, R. Perego, N. Tonellotto, and R. Venturini, "Fast ranking with additive ensembles of oblivious and non-oblivious regression trees," *ACM Trans. Inf. Syst.*, vol. 35, no. 2, pp. 15:1–15:31, 2016.

[28] B. C. Schäfer and Z. Wang, "High-level synthesis design space exploration: Past, present, and future," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 10, pp. 2628–2639, 2020.

[29] Q. Xu, T. Mytkowicz, and N. S. Kim, "Approximate computing: A survey," *IEEE Design Test*, vol. 33, no. 1, pp. 8–22, 2016.

[30] Q. Huang, R. Lian, A. Canis, J. Choi, R. Xi, S. Brown, and J. Anderson, "The effect of compiler optimizations on high-level synthesis for FPGAs," in *Annual International Symposium on Field-Programmable Custom Computing Machines*, 2013, pp. 89–96.

[31] Z.-H. Zhou, *Machine learning*. Springer Nature, 2021.

[32] P. R. Norvig and S. A. Intelligence, "Artificial intelligence: A modern approach," *Prentice Hall Upper Saddle River, NJ, USA: Rani, M., Nayak, R., & Vyas, OP (2015). An ontology-based adaptive personalized e-learning system, assisted by software agents on cloud storage. Knowledge-Based Systems*, vol. 90, pp. 33–48, 2002.

[33] S.-C. Wang, *Artificial Neural Network*. Springer US, 2003.

[34] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization." *Journal of machine learning research*, vol. 13, no. 2, 2012.

[35] F. Friedrichs and C. Igel, "Evolutionary tuning of multiple svm parameters," *Neurocomputing*, vol. 64, pp. 107–117, 2005.

[36] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," *Advances in neural information processing systems*, vol. 25, 2012.

[37] J. Wu, X.-Y. Chen, H. Zhang, L.-D. Xiong, H. Lei, and S.-H. Deng, "Hyperparameter optimization for machine learning models based on bayesian optimization," *Journal of Electronic Science and Technology*, vol. 17, no. 1, pp. 26–40, 2019.

[38] B. Shekar and G. Dagnew, "Grid search-based hyperparameter tuning and classification of microarray cancer data," in *2019 Second international conference on advanced computational and communication paradigms (ICACCP)*. IEEE, 2019, pp. 1–8.

[39] M. A. Amirabadi, M. H. Kahaei, and S. A. Nezamalhosseini, "Novel suboptimal approaches for hyperparameter tuning of deep neural network [under the shelf of optical communication]," *Physical Communication*, vol. 41, p. 101057, 2020.

[40] A. H. Victoria and G. Maragatham, "Automatic tuning of hyperparameters using bayesian optimization," *Evolving Systems*, vol. 12, no. 1, pp. 217–223, 2021.

[41] T. Aarrestad, V. Loncar, N. Ghielmetti, M. Pierini, S. Summers, J. Ngadiuba, C. Petersson, H. Linander, Y. Iiyama, G. Di Guglielmo, J. Duarte, P. Harris, D. Rankin, S. Jindariani, K. Pedro, N. Tran, M. Liu, E. Kreinar, Z. Wu, and D. Hoang, "Fast convolutional neural networks on fpgas with hls4ml," *Machine Learning: Science and Technology*, vol. 2, no. 4, 7 2021.

[42] T. Choudhary, V. Mishra, A. Goswami, and J. Sarangapani, "A comprehensive survey on model compression and acceleration," *Artificial Intelligence Review*, vol. 53, no. 7, pp. 5113—5155, October 2020.

[43] H. Qin, R. Gong, X. Liu, X. Bai, J. Song, and N. Sebe, "Binary neural networks: A survey," *Pattern Recognition*, vol. 105, p. 107281, 2020.

[44] C. Buciluǎ, R. Caruana, and A. Niculescu-Mizil, "Model compression," in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2006, pp. 535–541.

[45] G. E. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *CoRR*, vol. abs/1503.02531, 2015.

[46] T. Liang, J. Glossner, L. Wang, S. Shi, and X. Zhang, "Pruning and quantization for deep neural network acceleration: A survey," *Neurocomputing*, vol. 461, pp. 370–403, 2021.

[47] P. Ganesh, Y. Chen, X. Lou, M. A. Khan, Y. Yang, H. Sajjad, P. Nakov, D. Chen, and M. Winslett, "Compressing large-scale transformer-based models: A case study on BERT," *Transactions of the Association for Computational Linguistics*, vol. 9, pp. 1061–1080, 2021.

[48] C. Buciluundefined, R. Caruana, and A. Niculescu-Mizil, "Model compression," in *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Association for Computing Machinery, 2006, pp. 535—541.

[49] B. Hawks, J. Duarte, N. J. Fraser, A. Pappalardo, N. Tran, and Y. Umuroglu, "Ps and qs: Quantization-aware pruning for efficient low latency neural network inference," *Frontiers in Artificial Intelligence*, vol. 4, p. 676564, 2021.

[50] S. Francescato, S. Giagu, F. Riti, G. Russo, L. Sabetta, and F. Tortonesi, "Model compression and simplification pipelines for fast deep neural network inference in fpgas in hep," *The European Physical Journal C*, vol. 81, p. 969, 11 2021.

[51] R. S. Molina, L. G. Garcia, I. R. Morales, M. L. Crespo, G. Ramponi, S. Carrato, A. Cicuttin, and H. Perez, "Compression of NN-based pulse-shape discriminators in front-end electronics for particle detection," in *Lecture Notes in Electrical Engineering*. Springer International Publishing, 2022, pp. 93–99.

[52] J. Duarte, S. Han, P. Harris, S. Jindariani, E. Kreinar, B. Kreis, J. Ngadiuba, M. Pierini, R. Rivera, N. Tran *et al.*, "Fast inference of deep neural networks in fpgas for particle physics," *Journal of Instrumentation*, vol. 13, no. 07, p. P07027, 2018.

[53] G. Urban, K. J. Geras, S. E. Kahou, O. Aslan, S. Wang, R. Caruana, A. Mohamed, M. Philipose, and M. Richardson, "Do deep convolutional nets really need to be deep and convolutional?" *arXiv preprint arXiv:1603.05691*, 2016.

[54] M. W. Jeter, *Mathematical programming: an introduction to optimization*. Routledge, 2018.

[55] K. Deb, "Multi-objective optimization," in *Search methodologies*. Springer, 2014, pp. 403–449.

[56] C. A. C. Coello, G. B. Lamont, D. A. Van Veldhuizen *et al.*, *Evolutionary algorithms for solving multi-objective problems*. Springer, 2007, vol. 5.

[57] S. Alarie, C. Audet, A. E. Gheribi, M. Kokkolaras, and S. Le Digabel, "Two decades of blackbox optimization applications," *EURO Journal on Computational Optimization*, vol. 9, p. 100011, 2021.

[58] P. I. Frazier, "Bayesian optimization," in *Recent advances in optimization and modeling of contemporary problems*. Informs, 2018, pp. 255–278.

[59] W. Gan, Z. Ji, and Y. Liang, "Acquisition functions in bayesian optimization," in *2021 2nd International Conference on Big Data & Artificial Intelligence & Software Engineering (ICBASE)*. IEEE, 2021, pp. 129–135.

[60] S. Greenhill, S. Rana, S. Gupta, P. Vellanki, and S. Venkatesh, "Bayesian optimization for adaptive experimental design: A review," *IEEE Access*, vol. 8, pp. 13 937–13 948, 2020.

[61] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas, "Taking the human out of the loop: A review of bayesian optimization," *Proceedings of the IEEE*, vol. 104, no. 1, pp. 148–175, 2016.

[62] T. Bartz-Beielstein, J. Branke, J. Mehnen, and O. Mersmann, "Evolutionary algorithms," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 4, no. 3, pp. 178–195, 2014.

[63] S. Das, B. K. Panigrahi, and S. S. Pattnaik, "Nature inspired methods for multi-objective optimization," in *Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques*. IGI Global, 2010, pp. 95–108.

[64] H. Du, Z. Wang, W. Zhan, and J. Guo, "Elitism and distance strategy for selection of evolutionary algorithms," *IEEE Access*, vol. 6, pp. 44 531–44 541, 2018.

[65] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan, "A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii," in *International conference on parallel problem solving from nature*. Springer, 2000, pp. 849–858.

[66] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[67] K. D. Tran, "Elitist non-dominated sorting ga-ii (nsga-ii) as a parameter-less multi-objective genetic algorithm," in *Proceedings. IEEE SoutheastCon, 2005.*, 2005, pp. 359–367.

[68] A. Zhou, B.-Y. Qu, H. Li, S.-Z. Zhao, P. N. Suganthan, and Q. Zhang, "Multiobjective evolutionary algorithms: A survey of the state of the art," *Swarm and evolutionary computation*, vol. 1, no. 1, pp. 32–49, 2011.

[69] S. Verma, M. Pant, and V. Snasel, "A comprehensive review on nsga-ii for multi-objective combinatorial optimization problems," *IEEE Access*, vol. 9, pp. 57 757–57 791, 2021.

[70] K. Deb, "Multi-objective optimisation using evolutionary algorithms: an introduction," in *Multi-objective evolutionary optimisation for product design and manufacturing*. Springer, 2011, pp. 3–34.

[71] D. R. F. de Bulnes, Y. Maldonado, and L. Trujillo, "Development of multiobjective high-level synthesis for FPGAs," *Scientific Programming*, vol. 2020, pp. 7 095 048:1–7 095 048:25, 2020.

[72] Z. Zeng, R. Sedaghat, and A. Sengupta, "A novel framework of optimizing modular computing architecture for multi objective VLSI designs," in *2009 International Conference on Microelectronics-ICM*, 2009, pp. 328–331.

[73] Y. Ma, S. Roy, J. Miao, J. Chen, and B. Yu, "Cross-layer optimization for high speed adders: A pareto driven machine learning approach," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 12, pp. 2298–2311, 2018.

[74] D. Roy and A. Sengupta, "Low overhead symmetrical protection of reusable ip core using robust fingerprinting and watermarking during high level synthesis," *Future Generation Computer Systems*, vol. 71, pp. 89–101, 2017.

[75] L. Piccolboni, P. Mantovani, G. D. Guglielmo, and L. P. Carloni, "Broadening the exploration of the accelerator design space in embedded scalable platforms," in *High Performance Extreme Computing Conference (HPEC)*, 2017, pp. 1–7.

[76] R. Resmi and B. B. T. Sundari, "Allocation of optimal reconfigurable array using graph merging technique," in *2014 International Conference on Embedded Systems (ICES)*, 2014, pp. 49–54.

[77] D. H. Ram, M. Bhuvaneswari, and S. Logesh, "A novel evolutionary technique for multi-objective power, area and delay optimization in high level synthesis of datapaths," in *Annual Symposium on Computer Society VLSI*, 2011, pp. 290–295.

[78] A. Sengupta, R. Sedaghat, and P. Sarkar, "A multi structure genetic algorithm for integrated design space exploration of scheduling and allocation in high level synthesis for DSP kernels," *Swarm and Evolutionary Computation*, vol. 7, pp. 35–46, 2012.

[79] A. Sengupta, R. Sedaghat, and P. Sarkar, "Rapid exploration of integrated scheduling and module selection in high level synthesis for application specific processor design," *Microprocess. Microsyst.*, vol. 36, no. 4, pp. 303—-314, 2012.

[80] B. C. Schafer and K. Wakabayashi, "Design space exploration acceleration through operation clustering," *IEEE Transactions on computer-aided design of integrated circuits and systems*, vol. 29, no. 1, pp. 153–157, 2009.

[81] B. C. Schafer, T. Takenaka, and K. Wakabayashi, "Adaptive simulated annealer for high level synthesis design space exploration," in *2009 International Symposium on VLSI Design, Automation and Test*, 2009, pp. 106–109.

[82] C. Lattner and V. Adve, "LLVM: a compilation framework for lifelong program analysis amp; transformation," in *International Symposium on Code Generation and Optimization, 2004. CGO 2004.*, 2004, pp. 75–86.

[83] LLVM Developer Group, "Clang." [Online]. Available: https://clang.llvm.org

[84] L. Huang, D. Li, K. Wang, T. Gao, and A. Tavares, "A survey on performance optimization of high-level synthesis tools," *Journal of Computer Science and Technology*, vol. 35, pp. 697–720, 2020.

[85] J. Zhao, L. Feng, S. Sinha, W. Zhang, Y. Liang, and B. He, "COMBA: A comprehensive model-based analysis framework for high level synthesis of real applications," in *International Conference on Computer-Aided Design (ICCAD), pages=430–437, year=2017, doi = 10.1109/ICCAD.2017.8203809.*

[86] Y.-k. Choi and J. Cong, "HLS-based optimization and design space exploration for applications with variable loop bounds," in *International Conference on Computer-Aided Design (ICCAD)*, 2018, pp. 1–8.

[87] J. S. Monson and B. L. Hutchings, "Using source-level transformations to improve high-level synthesis debug and validation on FPGAs," in *International Symposium on Field-Programmable Gate Arrays*, 2015, pp. 5–8.

[88] C. Li, Y. Bi, Y. Benezeth, D. Ginhac, and F. Yang, "High-level synthesis for FPGAs: code optimization strategies for real-time image processing," *Real-Time Image Processing*, vol. 14, no. 3, pp. 701–712, 2018.

[89] R. Campos and J. M. Cardoso, "On data parallelism code restructuring for HLS targeting FPGAs," in *International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2021, pp. 144–151.

[90] J. de Fine Licht, M. Besta, S. Meierhans, and T. Hoefler, "Transformations of high-level synthesis codes for high-performance computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 5, pp. 1014—-1029, may 2021.

[91] A. C. Ferreira and J. M. Cardoso, "Graph-based code restructuring targeting HLS for FPGAs," in *International Symposium on Applied Reconfigurable Computing*, 2019, pp. 230–244.

[92] J. de Fine Licht, M. Besta, S. Meierhans, and T. Hoefler, "Transformations of high-level synthesis codes for high-performance computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 5, pp. 1014–1029, 2021.

[93] M. Quoc Hoang, P. Luu Nguyen, H. Viet Tran, H. Quan Nguyen, V. Thang Nguyen, and C. Vo-Le, "FPGA oriented compression of DNN using layer-targeted weights and activations quantization," in *Eighth International Conference on Communications and Electronics (ICCE)*, 2021, pp. 157–162.

[94] Q. Zhang, J. Cao, Y. Zhang, S. Zhang, Q. Zhang, and D. Yu, "FPGA implementation of quantized convolutional neural networks," in *International Conference on Communication Technology (ICCT)*, 2019, pp. 1605–1610.

[95] P. Bacchus, R. Stewart, and E. Komendantskaya, "Accuracy, training time and hardware efficiency trade-offs for quantized neural networks on FPGAs," in *International symposium on applied reconfigurable computing*, 2020, pp. 121–135.

[96] X. Xu, Q. Lu, T. Wang, Y. Hu, C. Zhuo, J. Liu, and Y. Shi, "Efficient hardware implementation of cellular neural networks with incremental quantization and early exit," *Journal on Emerging Technologies in Computing Systems*, vol. 14, no. 4, pp. 1–20, 2018.

[97] S. A. Cook and R. A. Reckhow, "Time-bounded random access machines," *Symposium on Theory of Computing*, vol. 7, no. 4, pp. 354–375, 1972.

[98] S. Fortune and J. Wyllie, "Parallelism in random access machines," in *Symposium on Theory of Computing*, 1978, pp. 114–118.

[99] P. B. Gibbons, "A more practical PRAM model," in *Symposium on Parallel Algorithms and Architectures, SPAA*, 1989, pp. 158–168.

[100] L. G. Valiant, "A bridging model for parallel computation," *Communications of the ACM*, vol. 33, no. 8, p. 103–111, 1990.

[101] M. Kechid and J. Myoupo, "Towards a more realistic bsp cost model," in *Eighth International Conference on High-Performance Computing in Asia-Pacific Region (HPCASIA'05)*, 2005, pp. 10–12.

[102] L. Valiant, "A bridging model for multi-core computing," *Journal of Computer and System Sciences*, vol. 77, no. 1, pp. 154–166, 2011.

[103] A. Goldchleger, A. Goldman, U. Hayashida, and F. Kon, "The implementation of the BSP parallel computing model on the InteGrade Grid Middleware," in *International Workshop on Middleware for Grid Computing*, ser. MGC '05, 2005, pp. 1—-6.

[104] V. Allombert, F. Gava, and J. Tesson, "Toward performance prediction for Multi-BSP programs in ML," in *International Conference on Algorithms and Architectures for Parallel Processing*, 2018, pp. 159–174.

[105] G. Trabes, V. Gil-Costa, M. Printista, and M. Marin, "Multi-BSP vs. BSP: A case of study for dell AMD multicores," in *Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, 2018, pp. 579–587.

[106] A. Savadi, M. Moradi, and H. Deldari, "Multi-DaC programming model: A variant of Multi-BSP model for divide-and-conquer algorithms," 2012, pp. 41—-46.

[107] M. Alaniz and S. Nesmachnow, "A semi-automatic approach for parallel problem solving using the Multi-BSP model," *Programming and Computer Software*, vol. 45, no. 8, pp. 517–531, 2019.

[108] Z. Zeng and X. Sun, "Electric vehicle regional management system based on the BSP model and multi-information fusion," *Systems Science & Control Engineering*, vol. 9, no. sup1, pp. 114–121, 2021.

[109] X. Zhao, M. Papagelis, A. An, B. X. Chen, J. Liu, and Y. Hu, "ZipLine: an optimized algorithm for the elastic bulk synchronous parallel model," *Machine Learning*, vol. 110, no. 10, pp. 2867–2903, 2021.

[110] K. Siddique, Z. Akhtar, H. Lee, W. Kim, and Y. Kim, "Toward bulk synchronous parallel-based machine learning techniques for anomaly detection in high-speed big data networks," *Symmetry*, vol. 9, no. 9, p. 197, 2017.

[111] X. Zhao, M. Papagelis, A. An, B. X. Chen, J. Liu, and Y. Hu, "Elastic bulk synchronous parallel model for distributed deep learning," in *International Conference on Data Mining, ICDM*, 2019, pp. 1504–1509.

[112] M. Amaris, D. Cordeiro, A. Goldman, and R. Y. De Camargo, "A simple bsp-based model to predict execution time in GPU applications," in *International Conference on High Performance Computing (HiPC)*, 2015, pp. 285–294.

[113] D. E. Culler, R. M. Karp, D. A. Patterson, A. Sahay, K. E. Schauser, E. E. Santos, R. Subramonian, and T. von Eicken, "LogP: Towards a realistic model of parallel computation," in *SIGPLAN Symposium on Principles & Practice of Parallel Programming (PPOPP)*, 1993, pp. 1–12.

[114] A. Nomura, H. Matsuba, and Y. Ishikawa, "Network performance model for TCP/IP based cluster computing," in *International Conference on Cluster Computing*, 2007, pp. 194–203.

[115] "LogGP: Incorporating long messages into the LogP model for parallel computation," *Parallel and Distributed Computing*, vol. 44, no. 1, pp. 71–79, 1997.

[116] C. Moritz and M. Frank, "LoGPG: Modeling network contention in message-passing programs," *Transactions on Parallel and Distributed Systems*, vol. 12, no. 4, pp. 404–415, 2001.

[117] T. Touyama and S. Horiguchi, "Performance evaluation of practical parallel computation model LogPQ," in *Proceedings Fourth International Symposium on Parallel Architectures, Algorithms, and Networks (I-SPAN'99)*, 1999, pp. 216–221.

[118] T. Kielmann, H. E. Bal, and K. Verstoep, "Fast measurement of LogP parameters for message passing platforms," in *Parallel and Distributed Processing IPDPS*, vol. 1800, 2000, pp. 1176–1183.

[119] L. Li, X. Zhang, J. Feng, and X. Dong, "mPlogP: A parallel computation model for heterogeneous multi-core computer," in *International Conference on Cluster, Cloud and Grid Computing*, 2010, pp. 679–684.

[120] G. Liu, Y. Wang, T. Zhao, J. Gu, and D. Li, "mHLogGP: A parallel computation model for cpu/gpu heterogeneous computing cluster," vol. 7513, 2012, pp. 217–224.

[121] J. L. Roda, F. Sande, C. Leon, J. A. Gonzalez, and C. Rodriguez, "The collective computing model," in *Euromicro Workshop on Parallel and Distributed*, 1999, pp. 19–26.

[122] S. Williams, A. Waterman, and D. A. Patterson, "Roofline: an insightful visual performance model for multicore architectures," *Commun. ACM*, vol. 52, no. 4, pp. 65–76, 2009.

[123] C. Yang, T. Kurth, and S. Williams, "Hierarchical roofline analysis for GPUs: Accelerating performance optimization for the NERSC-9 Perlmutter system," *Concurrency and Computation: Practice and Experience*, vol. 32, 2020.

[124] C. Yang, Y. Wang, T. Kurth, S. Farrell, and S. Williams, "Hierarchical roofline performance analysis for deep learning applications," in *Intelligent Computing*, 2021, pp. 473–491.

[125] B. da Silva, A. Braeken, E. H. D'Hollander, and A. Touhafi, "Performance modeling for FPGAs: Extending the roofline model with high-level synthesis tools," *International Journal of Reconfigurable Computing*, vol. 2013, pp. 428 078:1–428 078:10, 2013.

[126] B. da Silva, A. Braeken, E. H. D'Hollander, A. Touhafi, J. G. Cornelis, and J. Lemeire, "Comparing and combining GPU and FPGA accelerators in an image processing context," in *23rd International Conference on Field programmable Logic and Applications, FPL 2013, Porto, Portugal, September 2-4, 2013*, 2013, pp. 1–4.

[127] M. Siracusa, L. Di Tucci, M. Rabozzi, S. Williams, E. D. Sozzo, and M. D. Santambrogio, "A CAD-based methodology to optimize HLS code via the roofline model," in *International Conference on Computer-Aided Design*, 2020.

[128] N. Kapre and H. Patel, "Applying models of computation to opencl pipes for FPGA computing," in *International Workshop on OpenCL, IWOCL 2017*, 2017, pp. 9:1–9:4.

[129] M. Hora, V. Koncický, and J. Tetek, "Theoretical model of computation and algorithms for FPGA-based hardware accelerators," *CoRR*, vol. abs/1807.03611, 2018.

[130] E. Calore and S. F. Schifano, "Performance assessment of FPGAs as HPC accelerators using the FPGA empirical roofline," in *2021 31st International Conference on Field-Programmable Logic and Applications (FPL)*, 2021, pp. 83–90.

[131] T. Nguyen, S. Williams, M. Siracusa, C. MacLean, D. Doerfler, and N. J. Wright, "The performance and energy efficiency potential of FPGAs in scientific computing," in *Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*, 2020, pp. 8–19.

[132] E. A. Lee and D. G. Messerschmitt, "Synchronous data flow," *Proceedings of the IEEE*, vol. 75, no. 9, pp. 1235–1245, 1987.

[133] Y. Choi and J. Cong, "HLScope: High-level performance debugging for FPGA designs," in *Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2017, pp. 125–128.

[134] Y. Choi, P. Zhang, P. Li, and J. Cong, "HLScope+ : Fast and accurate performance estimation for FPGA HLS," in *International Conference on Computer-Aided Design, ICCAD*, 2017, pp. 691–698.

[135] K. Papadimitriou, A. Dollas, and S. Hauck, "Performance of partial reconfiguration in FPGA systems: A survey and a cost model," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 4, no. 4, pp. 36:1–36:24, 2011.

[136] S. Wang, Y. Liang, and W. Zhang, "FlexCL: An analytical performance model for OpenCL workloads on flexible FPGAs," in *Annual Design Automation Conference, DAC*, 2017, pp. 27:1–27:6.

[137] H. Mohammadi Makrani, F. Farahmand, H. Sayadi, S. Bondi, S. M. Pudukotai Dinakarrao, H. Homayoun, and S. Rafatirad, "Pyramid: Machine learning framework to estimate the optimal timing and resource usage of a high-level synthesis design," in *International Conference on Field Programmable Logic and Applications FPL*, 2019, pp. 397–403.

[138] F. Farahmand, A. Ferozpuri, W. Diehl, and K. Gaj, "Minerva: Automated hardware optimization tool," in *International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, 2017, pp. 1–8.

[139] Z.-k. Wang, B. He, W. Zhang, and S. Jiang, "A performance analysis framework for optimizing OpenCL applications on FPGAs," in *International Symposium on High Performance Computer Architecture, HPCA*, 2016, pp. 114–125.

[140] C. Larman and V. R. Basili, "Iterative and incremental developments. a brief history," *IEEE Computer*, vol. 36, no. 6, pp. 47–56, 2003.

[141] Y. Nasser, J. Lorandel, J.-C. Prévotet, and M. Hélard, "Rtl to transistor level power modeling and estimation techniques for FPGA and ASIC: A survey," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 3, pp. 479–493, 2021.

[142] J. Lorandel, J.-C. Prévotet, and M. Hélard, "Efficient modelling of FPGA-based IP blocks using neural networks," in *2016 International Symposium on Wireless Communication Systems (ISWCS)*, 2016, pp. 571–575.

[143] A. N. Tripathi and A. Rajawat, "An accurate and quick ann-based system-level dynamic power estimation model using LLVM IR profiling for FPGA designs," *Embedded Systems Letters*, vol. 12, no. 2, pp. 58–61, 2020.

[144] G. Verma, T. Singhal, R. Kumar, S. Chauhan, S. Shekhar, B. Pandey, and D. M. A. Hussain, "Heuristic and statistical power estimation model for FPGA based wireless systems," *Wireless Personal Communications*, vol. 106, no. 4, pp. 2087–2098, 2019.

[145] G. Verma, V. Khare, and M. Kumar, "More precise FPGA power estimation and validation tool (fpev_tool) for low power applications," *Wirel. Pers. Commun.*, vol. 106, no. 4, pp. 2237–2246, 2019.

[146] L. Deng, K. Sobti, and C. Chakrabarti, "Accurate models for estimating area and power of FPGA implementations," in *International Conference on Acoustics, Speech and Signal Processing*, 2008, pp. 1417–1420.

[147] J. Davis, E. Hung, J. Levine, E. Stott, P. Cheung, and G. Constantinides, "KAPow: High-accuracy, low-overhead online per-module power estimation for FPGA designs," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 11, pp. 2:1–2:22, 2018.

[148] Y. Liang, S. Wang, and W. Zhang, "FlexCL: A model of performance and power for opencl workloads on FPGAs," *Transactions on Computers*, vol. 67, no. 12, pp. 1750–1764, 2018.

[149] K. O'Neal, M. Liu, H. Tang, A. Kalantar, K. DeRenard, and P. Brisk, "HLSPredict: Cross platform performance prediction for FPGA high-level synthesis," in *International Conference on Computer-Aided Design (ICCAD)*, 2018, pp. 1–8.

[150] Z. Lin, J. Zhao, S. Sinha, and W. Zhang, "HL-Pow: A learning-based power modeling framework for high-level synthesis," in *Asia and South Pacific Design Automation Conference, ASP-DAC*, 2020, pp. 574–580.

[151] Z. Lin, Z. Yuan, J. Zhao, W. Zhang, H. Wang, and Y. Tian, "PowerGear: Early-stage power estimation in FPGA HLS via heterogeneous edge-centric GNNs," pp. 1341–1346.

[152] Y. S. Shao, B. Reagen, G. Wei, and D. Brooks, "Aladdin: A pre-RTL, power-performance accelerator simulator enabling large design space exploration of customized architectures," in *International Symposium on Computer Architecture, ISCA*, 2014, pp. 97–108.

[153] M. Makni, S. Niar, M. Baklouti, and M. Abid, "HAPE: A high-level area-power estimation framework for fpga-based accelerators," *Microprocessors and Microsystems*, vol. 63, pp. 11–27, 2018.

[154] J. Cong, W. Jiang, B. Liu, and Y. Zou, "Automatic memory partitioning and scheduling for throughput and power optimization," *ACM Transactions on Design Automation of Electronic Systems*, vol. 16, no. 2, pp. 15:1–15:25, 2011.

[155] P. N. Khanh, A. K. Singh, A. Kumar, and K. M. M. Aung, "Exploiting loop-array dependencies to accelerate the design space exploration with high level synthesis," in *Design, Automation & Test in Europe Conference & Exhibition, DATE*, 2015, pp. 157–162.

[156] L. Ferretti, J. Kwon, G. Ansaloni, G. D. Guglielmo, L. P. Carloni, and L. Pozzi, "Leveraging prior knowledge for effective design-space exploration in high-level synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 11, pp. 3736–3747, 2020.

[157] L. Piccolboni, P. Mantovani, G. D. Guglielmo, and L. P. Carloni, "COSMOS: Coordination of high-level synthesis and memory optimization for hardware accelerators," vol. 16, no. 5s, 2017.

[158] C. Lo and P. Chow, "Model-based optimization of high level synthesis directives," in *International Conference on Field Programmable Logic and Applications (FPL)*, 2016, pp. 1–10.

[159] J. Kwon and L. P. Carloni, "Transfer learning for design-space exploration with high-level synthesis," in *Workshop on Machine Learning for CAD*, 2020, pp. 163–168.

[160] S. Dai, Y. Zhou, H. Zhang, E. Ustun, E. F. Y. Young, and Z. Zhang, "Fast and accurate estimation of quality of results in high-level synthesis with machine learning," in *Annual International Symposium on Field-Programmable Custom Computing Machines, FCCM*, 2018, pp. 129–132.

[161] G. Zhong, A. Prakash, Y. Liang, T. Mitra, and S. Niar, "Lin-Analyzer: A high-level performance analysis tool for FPGA-based accelerators," in *Annual Design Automation Conference, DAC, Austin, TX, USA, June 5-9*, 2016, pp. 136:1–136:6.

[162] A. Bannwart Perina, J. Becker, and V. Bonato, "Lina: Timing-constrained high-level synthesis performance estimator for fast DSE," in *2019 International Conference on Field-Programmable Technology (ICFPT)*, 2019, pp. 343–346.

[163] G. Zhong, A. Prakash, S. Wang, Y. Liang, T. Mitra, and S. Niar, "Design space exploration of FPGA-based accelerators with multi-level parallelism," in *Design, Automation & Test in Europe Conference & Exhibition, DATE*, 2017, pp. 1141–1146.

[164] L. Ferretti, G. Ansaloni, and L. Pozzi, "Cluster-based heuristic for high level synthesis design space exploration," *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 1, pp. 35–43, 2021.

[165] L. Ferretti, G. Ansaloni, and L. Pozzi, "Lattice-traversing design space exploration for high level synthesis," in *International Conference on Computer Design (ICCD)*, 2018, pp. 210–217.

[166] J. Zhao, L. Feng, S. Sinha, W. Zhang, Y. Liang, and B. He, "Performance modeling and directives optimization for high level synthesis on FPGA," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 7, pp. 1428–1441, 2019.

[167] N. Wu, Y. Xie, and C. Hao, "IronMan: GNN-assisted design space exploration in high-level synthesis via reinforcement learning," in *GLSVLSI '21: Great Lakes Symposium on VLSI 2021, Virtual Event, USA, June 22-25, 2021*, 2021, pp. 39–44.

[168] M. Siracusa, E. Delsozzo, M. Rabozzi, L. Di Tucci, S. Williams, D. Sciuto, and M. D. Santambrogio, "A comprehensive methodology to optimize FPGA designs via the roofline model," *Transactions on Computers*, pp. 1–1, 2021.

[169] S. W. Nabi and W. Vanderbauwhede, "FPGA design space exploration for scientific HPC applications using a fast and accurate cost model based on roofline analysis," *Journal of Parallel and Distributed Computing*, vol. 133, pp. 407–419, 2019.

[170] R. Tessier and H. Giza, "Balancing logic utilization and area efficiency in FPGAs," in *Field-Programmable Logic and Applications, The Roadmap to Reconfigurable Computing, 10th International Workshop*, vol. 1896, 2000, pp. 535–544.

[171] D. Koeplinger, R. Prabhakar, Y. Zhang, C. Delimitrou, C. Kozyrakis, and K. Olukotun, "Automatic generation of efficient accelerators for reconfigurable hardware," in *Annual International Symposium on Computer Architecture, ISCA*, 2016, pp. 115–127.

[172] A. Sohrabizadeh, C. H. Yu, M. Gao, and J. Cong, "AutoDSE: Enabling software programmers to design efficient FPGA accelerators," 2021.

[173] Q. Gautier, A. Althoff, C. L. Crutchfield, and R. Kastner, "Sherlock: A multi-objective design space exploration framework," *ACM Transactions on Design Automation of Electronic Systems*, vol. 27, no. 4, 2022.

[174] B. C. Schafer and K. Wakabayashi, "Machine learning predictive modelling high-level synthesis design space exploration," *IET computers & digital techniques*, vol. 6, no. 3, pp. 153–159, 2012.

[175] F. Ferrandi, P. L. Lanzi, D. Loiacono, C. Pilato, and D. Sciuto, "A multi-objective genetic algorithm for design space exploration in high-level synthesis," in *2008 IEEE Computer Society Annual Symposium on VLSI*, 2008, pp. 417–422.

[176] C. Pilato, A. Tumeo, G. Palermo, F. Ferrandi, P. L. Lanzi, and D. Sciuto, "Improving evolutionary exploration to area-time optimization of fpga designs," *Journal of Systems Architecture*, vol. 54, no. 11, pp. 1046–1057, 2008.

[177] M. Yu, S. Huang, and D. Chen, "Chimera: A hybrid machine learning driven multi-objective design space exploration tool for fpga high-level synthesis," 2022.

[178] A. Mehrabi, A. Manocha, B. C. Lee, and D. J. Sorin, "Bayesian optimization for efficient accelerator synthesis," vol. 18, no. 1, 2021.

[179] H. Mohammadi Makrani, H. Sayadi, T. Mohsenin, S. Rafatirad, A. Sasan, and H. Homayoun, "XPPE: cross-platform performance estimation of hardware accelerators using machine learning," in *Asia and South Pacific Design Automation Conference, ASPDAC*, 2019, pp. 727–732.

[180] P. Meng, A. Althoff, Q. Gautier, and R. Kastner, "Adaptive threshold non-pareto elimination: Re-thinking machine learning for system level design space exploration on FPGAs," in *Design, Automation Test in Europe Conference Exhibition (DATE)*, 2016, pp. 918–923.

[181] S. Xu, S. Liu, Y. Liu, A. Mahapatra, M. Villaverde, F. Moreno, and B. C. Schäfer, "Design space exploration of heterogeneous MPSoCs with variable number of hardware accelerators," *Microprocessors and Microsystems*, vol. 65, pp. 169–179, 2019.

[182] Y. Nasser, J. Prévotet, and M. Hélard, "Power modeling on FPGA: a neural model for RT-level power estimation," in *International Conference on Computing Frontiers, CF*, 2018, pp. 309–313.

[183] S. Liu, F. C. Lau, and B. C. Schafer, "Accelerating FPGA prototyping through predictive model-based HLS design space exploration," in *Annual Design Automation Conference, DAC*, 2019, p. 97.

[184] S. Xu and B. C. Schafer, "Approximating behavioral HW accelerators through selective partial extractions onto synthesizable predictive models," in *International Conference on Computer-Aided Design (ICCAD)*, 2019, pp. 1–8.

[185] A. S. B. Lopes and 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)*, 2020, pp. 1–6.

[186] C. Du and Y. Yamaguchi, "High-level synthesis design for stencil computations on FPGA with high bandwidth memory," *MDPI Electronics*, vol. 9, no. 8, 2020.

[187] M. Karp, A. Podobas, N. Jansson, T. Kenter, C. Plessl, P. Schlatter, and S. Markidis, "High-performance spectral element methods on field-programmable gate arrays : Implementation, evaluation, and future projection," in *International Parallel and Distributed Processing Symposium, IPDPS*, 2021, pp. 1077–1086.

[188] K. Nagasu, K. Sano, F. Kono, and N. Nakasato, "FPGA-based tsunami simulation: Performance comparison with GPUs, and roofline model for scalability analysis," *Parallel and Distributed Computing*, vol. 106, pp. 153–169, 2017.

[189] C. Du, I. Firmansyah, and Y. Yamaguchi, "FPGA-based computational fluid dynamics simulation architecture via high-level synthesis design method," in *Applied Reconfigurable Computing. Architectures, Tools, and Applications*, vol. 12083, 2020, pp. 232–246.

[190] E. Reggiani, G. Natale, C. Moroni, and M. D. Santambrogio, "An FPGA-based acceleration methodology and performance model for iterative stencils," in *International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2018, pp. 115–122.

[191] M. Manuel, A. Kreddig, S. Conrady, N. Anh Vu Doan, and W. Stechele, "Model-based design space exploration for approximate image processing on FPGA," in *Nordic Circuits and Systems Conference, NorCAS 2020, Oslo*, 2020, pp. 1–7.

[192] E. Reggiani, M. Rabozzi, A. M. Nestorov, A. Scolari, L. Stornaiuolo, and M. Santambrogio, "Pareto optimal design space exploration for accelerated CNN on FPGA," in *International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2019, pp. 107–114.

[193] P. Gysel, V. Akella, and S. Ghiasi, "Design space exploration of FPGA-based deep convolutional neural networks," 2016, pp. 575–580.

[194] J. Xu, Z. Liu, J. Jiang, Y. Dou, and S. Li, "CaFPGA: An automatic generation model for CNN accelerator," *Microprocessors and Microsystems*, vol. 60, pp. 196–206, 2018.

[195] J. Shan, M. T. Lazarescu, J. Cortadella, L. Lavagno, and M. R. Casu, "CNN-on-AWS: Efficient allocation of multikernel applications on multi-fpga platforms," *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 2, pp. 301–314, 2021.

[196] S. O. Ayat, M. Khalil-Hani, and A. Rahman, "Optimizing FPGA-based CNN accelerator for energy efficiency with an extended roofline model," *Turkish Journal of Electrical Engineering and Computer Sciences*, vol. 26, pp. 919–935, 2018.

[197] L. Xie, X. Fan, W. Cao, and L. Wang, "High throughput CNN accelerator design based on FPGA," in *2018 International Conference on Field-Programmable Technology (FPT)*, 2018, pp. 274–277.

[198] C. Park, S. Park, and C. S. Park, "Roofline-model-based design space exploration for dataflow techniques of CNN accelerators," *IEEE Access*, vol. 8, pp. 172 509–172 523, 2020.

[199] Y. Ma, Y. Cao, S. B. K. Vrudhula, and J. Seo, "Performance modeling for CNN inference accelerators on FPGA," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 4, pp. 843–856, 2020.

[200] S. Conrady, A. Kreddig, M. Manuel, N. A. V. Doan, and W. Stechele, "Model-based design space exploration for fpga-based image processing applications employing parameterizable approximations," *Microprocessors and Microsystems*, vol. 87, p. 104386, 2021.

[201] T. Geng, T. Wang, A. Li, X. Jin, and M. C. Herbordt, "A scalable framework for acceleration of CNN training on deeply-pipelined FPGA clusters with weight and workload balancing," *CoRR*, vol. abs/1901.01007, 2019.

[202] W. Zhao, H. Fu, W. Luk, T. Yu, S. Wang, B. Feng, Y. Ma, and G. Yang, "F-CNN: An FPGA-based framework for training convolutional neural networks," in *International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, 2016, pp. 107–114.

[203] Y.-C. Lin, B. Zhang, and V. Prasanna, "HP-GNN: Generating high throughput GNN training implementation on CPU-FPGA heterogeneous platform," 2022, pp. 123—-133.

[204] A. Ghaffari and Y. Savaria, "CNN2Gate: An implementation of convolutional neural networks inference on FPGAs with automated design space exploration," *MDPI Electronics*, vol. 9, no. 12, p. 2200, 2020.

[205] S. I. Venieris and C.-S. Bouganis, "fpgaConvNet: Automated mapping of convolutional neural networks on FPGAs," in *International symposium on field-programmable gate arrays*, 2017, pp. 291–292.

[206] Y. Chen, J. He, X. Zhang, C. Hao, and D. Chen, "Cloud-DNN: An open framework for mapping DNN models to cloud FPGAs," in *International symposium on field-programmable gate arrays*, 2019, pp. 73–82.

[207] A. Biondi, A. Balsini, M. Pagani, E. Rossi, M. Marinoni, and G. Buttazzo, "A framework for supporting real-time applications on dynamic reconfigurable FPGAs," in *Real-Time Systems Symposium (RTSS)*, 2016, pp. 1–12.

[208] J. Mu, W. Zhang, H. Liang, and S. Sinha, "A collaborative framework for FPGA-based CNN design modeling and optimization," in *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*, 2018, pp. 139–1397.

[209] O. Reiche, M. A. Özkan, R. Membarth, J. Teich, and F. Hannig, "Generating FPGA-based image processing accelerators with hipacc: (invited paper)," in *International Conference on Computer-Aided Design (ICCAD)*, 2017, pp. 1026–1033.

[210] C. H. Yu, P. Wei, M. Grossman, P. Zhang, V. Sarker, and J. Cong, "S2FA: An accelerator automation framework for heterogeneous computing in datacenters," in *Design Automation Conference (DAC)*, 2018, pp. 153:1–153:6.

[211] P. Xu, X. Zhang, C. Hao, Y. Zhao, Y. Zhang, Y. Wang, C. Li, Z. Guan, D. Chen, and Y. Lin, "AutoDNNchip: An automated DNN chip predictor and builder for both FPGAs and ASICs," in *International Symposium on Field-Programmable Gate Arrays*, 2020, pp. 40–50.

[212] J. Jobson, "Multiple linear regression," in *Applied multivariate data analysis*. Springer, 1991, pp. 219–398.

[213] G. K. Uyanık and N. Güler, "A study on multiple linear regression analysis," *Procedia - Social and Behavioral Sciences*, vol. 106, pp. 234–240, 2013, 4th International Conference on New Horizons in Education.

[214] W. T. Ambrosius, *Topics in biostatistics*. Springer Science & Business Media, 2007, vol. 404.

[215] L. E. Eberly, "Multiple linear regression," *Topics in Biostatistics*, pp. 165–187, 2007.

[216] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[217] X. Gao, J. Wickerson, and G. A. Constantinides, "Automatically optimizing the latency, area, and accuracy of c programs for high-level synthesis," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2016, pp. 234–243.

[218] X. Inc., "Vitis high-level synthesis user guide. UG1399," 2021.

[219] M. G. Genton, "Classes of kernels for machine learning: A statistics perspective," *J. Mach. Learn. Res.*, vol. 2, pp. 299—312, mar 2002.

[220] J. Močkus, "On bayesian methods for seeking the extremum," in *Optimization techniques IFIP technical conference*. Springer, 1975, pp. 400–404.

[221] D. R. Jones, M. Schonlau, and W. J. Welch, "Efficient global optimization of expensive black-box functions," *Journal of Global optimization*, vol. 13, no. 4, pp. 455–492, 1998.

[222] J. Blank and K. Deb, "Pymoo: Multi-objective optimization in python," *IEEE Access*, vol. 8, pp. 89 497–89 509, 2020.

[223] B. C. Schäfer, "Probabilistic multiknob high-level synthesis design space exploration acceleration," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, pp. 394–406, 2016.

[224] L. G. G. Ordóñez, R. S. Molina, I. R. M. Argueta, M. L. Crespo, A. Cicuttin, S. Carrato, G. Ramponi, H. E. P. Figueroa, and M. G. B. Escobar, "Pulse shape discrimination for online data acquisition in water cherenkov detectors based on FPGA/SoC," in *Proceedings of 37th International Cosmic Ray Conference PoS(ICRC2021)*. Sissa Medialab, jul 2021.

[225] L. G. Garcia, R. S. Molina, M. L. Crespo, S. Carrato, G. Ramponi, A. Cicuttin, I. R. Morales, and H. Perez, "Muon–electron pulse shape discrimination for water cherenkov detectors based on fpga/soc," *Electronics*, vol. 10, no. 3, 2021.

[226] T. K. Gaisser, R. Engel, and E. Resconi, *Cosmic rays and particle physics*. Cambridge University Press, 2016.

[227] P. Abbon, E. Albrecht, V. Y. Alexakhin, Y. Alexandrov, G. Alexeev, M. Alekseev, A. Amoroso, H. Angerer, V. Anosov, B. Badełek *et al.*, "The compass experiment at cern," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 577, no. 3, pp. 455–518, 2007.

[228] E. Mace, J. Ward, and C. Aalseth, "Use of neural networks to analyze pulse shape data in low-background detectors," *Journal of Radioanalytical and Nuclear Chemistry*, pp. 1–8, 07 2018.

[229] J. Griffiths, S. Kleinegesse, D. Saunders, R. Taylor, and A. Vacheret, "Pulse shape discrimination and exploration of scintillation signals using convolutional neural networks," 2018.

[230] P. Holl, L. Hauertmann, B. Majorovits, O. Schulz, M. Schuster, and A. J. Zsigmond, "Deep learning based pulse shape discrimination for germanium detectors," *The European Physical Journal C*, vol. 79, no. 6, May 2019.

[231] D. Droz, A. Tykhonov, and X. Wu, "Neural Networks for Electron Identification with DAMPE," in *Proceedings of 36th International Cosmic Ray Conference — PoS(ICRC2019)*, vol. 358, 2019, p. 064.

[232] M. Berthold and F. Höppner, "On clustering time series using euclidean distance and pearson correlation. arxiv 2016," *arXiv preprint arXiv:1601.02213*.

[233] F. J. Provost and T. Fawcett, "Analysis and visualization of classifier performance with nonuniform class and cost distributions," 1997.

[234] L. Omar and I. P. Ivrissimtzis, "Using theoretical ROC curves for analysing machine learning binary classifiers," *Pattern Recognit. Lett.*, vol. 128, pp. 447–451, 2019.

[235] K. Feng, H. Hong, K. Tang, and J. Wang, "Decision making with machine learning and ROC curves," *CoRR*, vol. abs/1905.02810, 2019.

[236] A. Suarez, R. S. Molina, G. Ramponi, R. Petrino, L. Bollati, and D. Sequeiros, "Pest detection and classification to reduce pesticide use in fruit crops based on deep neural networks and image processing," in *2021 XIX Workshop on Information Processing and Control (RPIC)*. IEEE, nov 2021.

[237] Molina, R. S.; Carrer. V.; Ballina; M., Crespo, M. L.; Bollati, L.; Sequeiro, D.; Marsi, S. and Ramponi, G., "Ml-based classifier for precision agriculture on embedded systems," in *International Conference on Applications in Electronics Pervading Industry, Environment and Society*, 2022.

[238] A. Albanese, M. Nardello, and D. Brunelli, "Automated pest detection with dnn on the edge for precision agriculture," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 11, no. 3, pp. 458–467, 2021.

[239] L. Liu, R. Wang, C. Xie, P. Yang, F. Wang, S. Sudirman, and W. Liu, "Pestnet: An end-to-end deep learning approach for large-scale multi-class pest detection and classification," *IEEE Access*, vol. 7, pp. 45 301–45 312, 2019.

[240] N. T. Nam and P. D. Hung, "Pest detection on traps using deep convolutional neural networks," ser. ICCCV '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 33–38.

[241] M. Karar, F. Alsunaydi, S. Albusaymi, and S. Alotaibi, "A new mobile application of agricultural pests recognition using deep learning in cloud computing system," *AEJ - Alexandria Engineering Journal*, vol. 60, pp. 4423–4432, March 2021.

[242] C.-J. Chen, Y.-Y. Huang, Y.-S. Li, C.-Y. Chang, and Y.-M. Huang, "An AIoT based smart agricultural system for pests detection," *IEEE Access*, vol. 8, pp. 1–1, January 2020.

[243] V. K. Quy, N. V. Hau, D. V. Anh, N. M. Quy, N. T. Ban, S. Lanza, G. Randazzo, and A. Muzirafuti, "Iot-enabled smart agriculture: Architecture, applications, and challenges," *Applied Sciences*, vol. 12, no. 7, 2022.

[244] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He, "A comprehensive survey on transfer learning," *Proceedings of the IEEE*, vol. 109, no. 1, pp. 43–76, 2020.

[245] X. Wu, C. Zhan, Y.-K. Lai, M.-M. Cheng, and J. Yang, "IP102: A large-scale benchmark dataset for insect pest recognition," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 8779–8788.

[246] R. Wang, L. Liu, C. Xie, P. Yang, R. Li, and M. Zhou, "AgriPest: A large-scale domain-specific benchmark dataset for practical agricultural pest detection in the wild," *Sensors*, vol. 21, February 2021.

[247] Q.-J. Wang, S.-Y. Zhang, S.-F. Dong, G.-C. Zhang, J. Yang, R. Li, and H.-Q. Wang, "Pest24: A large-scale very small object data set of agricultural pests for multi-target detection," *Computers and Electronics in Agriculture*, vol. 175, p. 105585, 2020.

[248] R. Molina, F. Loor, V. Gil-Costa, F. M. Nardini, R. Perego, and S. Trani, "Efficient traversal of decision tree ensembles with FPGAs," *Journal of Parallel and Distributed Computing*, vol. 155, pp. 38–49, sep 2021.

[249] V. Gil-Costa, F. Loor, R. Molina, F. M. Nardini, R. Perego, and S. Trani, "Ensemble model compression for fast and energy-efficient ranking on FPGAs," in *Lecture Notes in Computer Science*. Springer International Publishing, 2022, pp. 260–273.

[250] G. J. Kowalski, *Information retrieval systems: theory and implementation*. springer, 2007, vol. 1.

[251] F. Lettich, C. Lucchese, F. M. Nardini, S. Orlando, R. Perego, N. Tonellotto, and R. Venturini, "Multicore/manycore parallel traversal of large forests of regression trees," in *2017 International Conference on High Performance Computing Simulation (HPCS)*, 2017, pp. 915–915.

[252] F. Lettich, C. Lucchese, F. M. Nardini, S. Orlando, R. Perego, N. Tonellotto, and R. Venturini, "Parallel traversal of large ensembles of decision trees," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 9, pp. 2075–2089, 2019.

[253] I. Segalovich, "Machine learning in search quality at Yandex," Presentation at the industry track of the 33rd Annual ACM SIGIR Conference, 2010.

[254] A. Shchekalev, "Using gpus to accelerate learning to rank," in *Proceedings of the NVIDIA GTC-GPU Technology Conference*, 2014.

[255] D. Sorokina and E. Cantu-Paz, "Amazon search: The joy of ranking products," in *Proc. ACM SIGIR*, 2016, pp. 459–460.

[256] X. He, J. Pan, O. Jin, T. Xu, B. Liu, T. Xu, Y. Shi, A. Atallah, R. Herbrich, S. Bowers, and J. Q. Candela, "Practical Lessons from Predicting Clicks on Ads at Facebook," in *Proc. 8th International Workshop on Data Mining for Online Advertising*, 2014, pp. 5:1–5:9.

[257] T.-Y. Liu, "Learning to rank for information retrieval," *Foundations and Trends in Information Retrieval*, vol. 3, no. 3, pp. 225–331, March 2009.

[258] K. Järvelin and J. Kekäläinen, "Cumulated gain-based evaluation of ir techniques," *ACM Transactions on Information Systems (TOIS)*, vol. 20, no. 4, pp. 422–446, 2002.

[259] J. Blank and K. Deb, "A running performance metric and termination criterion for evaluating evolutionary multi-and many-objective optimization algorithms," in *2020 IEEE Congress on Evolutionary Computation CEC*. IEEE, 2020, pp. 1–8.

[260] K. Deb, K. Sindhya, and T. Okabe, "Self-adaptive simulated binary crossover for real-parameter optimization," in *Proceedings of the 9th annual conference on genetic and evolutionary computation*, 2007, pp. 1187–1194.

[261] K. Manev, A. Vaishnav, and D. Koch, "Unexpected diversity: Quantitative memory analysis for zynq ultrascale+ systems," in *2019 International Conference on Field-Programmable Technology (ICFPT)*. IEEE, 2019, pp. 179–187.