Received 22 July 2022, accepted 16 August 2022, date of publication 23 August 2022, date of current version 1 September 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3201107

SURVEY

High-Level Synthesis Hardware Design for FPGA-Based Accelerators: Models, Methodologies, and Frameworks

ROMINA SOLEDAD MOLINA[1,3,4], (Student Member, IEEE), VERONICA GIL-COSTA[2], MARÍA LIZ CRESPO[3], AND GIOVANNI RAMPONI[1], (Life Senior Member, IEEE)

[1]Dipartimento di Ingegneria e Architettura (DIA), Università degli Studi di Trieste, 34127 Trieste, Italy
[2]CONICET, Universidad Nacional de San Luis, San Luis D5700HHW, Argentina
[3]Multidisciplinary Laboratory (MLab), The Abdus Salam International Centre for Theoretical Physics, 34151 Trieste, Italy
[4]Departamento de electrónica, Universidad Nacional de San Luis, San Luis D5700HHW, Argentina

Corresponding author: Romina Soledad Molina (rominasoledad.molina@phd.units.it)

The work of Romina Soledad Molina was supported by the University of Trieste and the Abdus Salam International Centre for Theoretical Physics.

ABSTRACT Hardware accelerators based on field programmable gate array (FPGA) and system on chip (SoC) devices have gained attention in recent years. One of the main reasons is that these devices contain reconfigurable logic, which makes them feasible for boosting the performance of applications. High-level synthesis (HLS) tools facilitate the creation of FPGA code from a high level of abstraction using different directives to obtain an optimized hardware design based on performance metrics. However, the complexity of the design space depends on different factors such as the number of directives used in the source code, the available resources in the device, and the clock frequency. Design space exploration (DSE) techniques comprise the evaluation of multiple implementations with different combinations of directives to obtain a design with a good compromise between different metrics. This paper presents a survey of models, methodologies, and frameworks proposed for metric estimation, FPGA-based DSE, and power consumption estimation on FPGA/SoC. The main features, limitations, and trade-offs of these approaches are described. We also present the integration of existing models and frameworks in diverse research areas and identify the different challenges to be addressed.

INDEX TERMS Computing models, design space exploration, field programmable gate array (FPGA), system on chip (SoC), power consumption.

I. INTRODUCTION

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) [1], [2], [3], [4] 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 [5] have been proposed by vendors and academics such as Vivado HLS [6], formerly AutoPilot [7], Intel HLS [8], LegUp [9], Bambu [10], and others [5]. 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 the use of code restructuring techniques in the original algorithm [11]. This is a time-consuming task for algorithm designers, who want to take advantage of the inherent characteristics of these reconfigurable technologies.

The associate editor coordinating the review of this manuscript and approving it for publication was Vincenzo Conti.

HLS tools support C/C++, SystemC, and OpenCL [12] codes to generate the final RTL code. 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 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 increases exponentially as the designer applies more directives, and the program has more complex code structures. The generated hardware is directly associated with the applied directives, but sometimes applying and tuning directives requires a considerable endeavour 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 software developer. The actual trend in parallel computer architectures demonstrates progress toward hybrid architectures combining namely many cores, superscalars, single instruction/multiple data (SIMD), hardware accelerators, and on-chip communication systems, among others, which require handling computations and data locality at several levels to achieve suitable performance [13].

Using computing models, and also 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. In this survey, a model is an abstraction that represents a simplified system. A methodology describes the steps involved in the process for systematically solving a problem. A framework provides the structure needed in the form of a template or conceptual scheme to simplify the elaboration of a task.

A. CONTRIBUTION

In this paper, we present a thorough analysis of the computing models, methodologies, and frameworks proposed for reconfigurable hardware accelerators based on FPGA. We compare their main features, including the inputs, outputs, and techniques employed for their development. Then, we show how these approaches for FPGA/SoC can be applied in different research fields, exposing their benefits in improving the design process and productivity. Consequently, the reader will become more confident about the fundamental and technical aspects of the computing models, methodologies, and frameworks designed for FPGA/SoC, acquiring a clear idea of the main parameters required by each one. We highlight the importance of having simple approaches with few parameters, such as those proposed for other parallel architectures, so that they have a greater scope and can be widely used. Based on this literature review, the FPGA developer can select the approach that best suits the application, hardware architecture, and programming skills.

Some survey articles are available in the literature for FPGA-based reconfigurable hardware. Schafer and Wang [14] divide HLS DSE techniques into two main groups: synthesis-based and model-based. In addition to this classification, a third group appears including DSE synthesis-based and supervised learning. According to [15], HLS DSE can be developed using model-based and model-free techniques. Model-based techniques are composed of tools and methodologies that use analytical models, whereas model-free techniques include approaches where the HLS tool is treated as a black box. A survey of automatic high-level code deployment for HLS tools and toolchains is presented in [16]. The authors analyze commercial HLS tools, academic HLS tools, HLS code generation tools, domain-specific language tools for HLS, dataflow HLS tools, and automatic code deployment tools (including automated DSE). Yehya *et al.* [17] focus on power consumption. They classify different estimation techniques as analytical, table-based, polynomial-based, and neural networks. The work in [18] analyzes different performance and power estimation models for CPU, GPU, and FPGA. Moreover, reconfigurable architectures can be categorized as coarse-grained and fine-grained according to [19], [20]. In this work, we focus on FPGA and FPGA/SoC architectures included in the last category.

To the best of our knowledge, **there is no previous work that jointly**:

- describes the models, methodologies, and frameworks developed for the estimation of metrics, FPGA-based DSE, and power consumption estimation on FPGA/SoC,
- shows their application in different research areas,
- analyzes the challenges to be addressed to widely use them for FPGA/SoC,
- compares them with the commonly used parallel computing models for CPU, GPU, and multicore processors.

B. METHODOLOGY

This survey is conducted by collecting the latest contributions, focusing on the models, methodologies, and frameworks for FPGA-based devices. The paper collection process has been performed mainly using models, methodologies, FPGA/SoC, parallel computing models, DSE, and Pareto-optimal design keywords in well-known scientific databases such as IEEE Xplore, Scopus, Web of Science, ScienceDirect, arXiv, and Directory of Open Access Journals (DOAJ). The collected contributions are from the last six

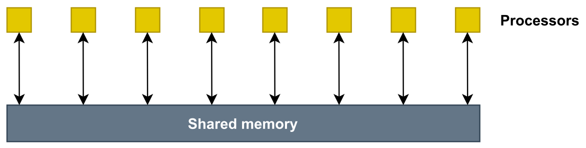

FIGURE 1. PRAM model. Different processors execute read and write operations in a shared memory.

years (2016–2022) and have been selected based on the topics addressed in this survey. Several papers published before 2016 have been considered because of their contributions to the current literature.

C. OUTLINE

The remainder of this paper is organized as follows. Section II briefly presents the most widely used parallel computing models for CPU, GPU, and multicore processors. Section III introduces the FPGA-based reconfigurable hardware accelerator architectures, hardware/software co-design, DSE and metrics, and the techniques to improve latency, area, and power for this technology. In Section IV, we describe previous works on models, methodologies, and frameworks proposed for FPGA/SoC according to their main features: metrics estimation (IV-A), FPGA-based DSE (IV-B), and power consumption estimation (IV-C); and in Section IV-D, we present a summary and discussion. The integration of models and frameworks for FPGA-based reconfigurable hardware accelerators in different research fields is exposed in Section V. Challenges are analyzed in Section VI. Finally, conclusions are presented in Section VII.

II. 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 small number of operations, and a high degree of parallelism are key 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. In addition, we present some examples of their application in different architectures.

A. RANDOM ACCESS MACHINE AND PARALLEL RANDOM ACCESS MACHINE

The random access machine (RAM) model is proposed in [21] for sequential algorithms. It is composed of a memory, control unit, processor, and program. In 1978, Fortune and Wyllie proposed the parallel random access machine (PRAM) model [22] 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. 1. In this scenario, one processor P can execute one operation (arithmetic, memory access, or logic) within one single 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) are introduced to handle read/write operations in a shared memory model [23].

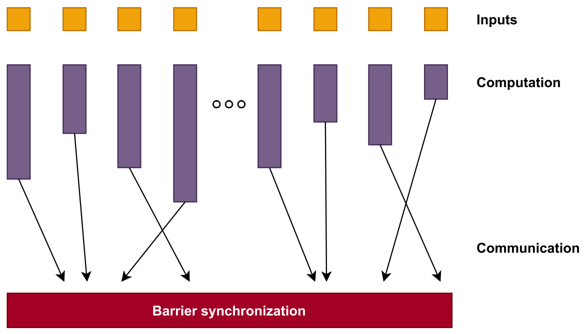

FIGURE 2. Superstep of the BSP model.

B. BULK SYNCHRONOUS PARALLEL MODEL

The bulk synchronous parallel model (BSP) [24] 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. 2 shows the workflow of the BSP model.

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.1

$$G = \frac{Op_{local}}{W_{sec}} \tag{1}$$

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

The sum of G and L is the superstep cost. The former represents the number of maximum local computations executed on parallel processors. The latter represents a cost composed of the cost of the communications plus the synchronization at the end of the superstep.

The multi-BSP model [25] extends the BSP to multicore architectures by considering the architecture as a tree with d

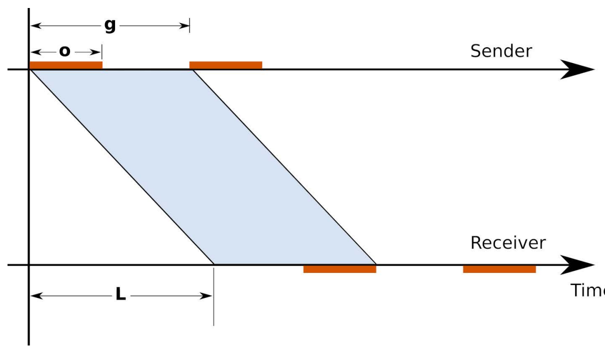

FIGURE 3. LogP model, based on [37]. From a local point of view, for one processor (*P*), *g* represents the gap between messages, *o* is the communication overhead, and *L* is the communication delay.

leaves. This 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 [26], [27], [28], [29], [30], [31], [32]. The results presented in [33] demonstrate the feasibility of the BSP-based machine learning (ML) computing model in the field of intrusion detection. An elastic BSP for relaxing the synchronization stage in the context of distributed deep learning is presented in [34]. The authors focus on the data parallelism approach, in which weight synchronization during training is crucial. The BSP is adapted for CUDA applications in [35]. This BSP for the CUDA model allows the prediction of execution times for a single kernel function on the GPU. This proposal focuses on a number of computational and communication steps, but removes synchronization at the end of each step.

C. LogP MODEL

The LogP model [36] describes a parallel machine using four main parameters: communication delay (L), communication overhead (o), gap between each message (g, from a local point of view), and the number of processors (P). A graphical representation of the different parameters is presented in Fig. 3. 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, is based on message passing, and can simulate a BSP model.

Different variants of LogP, such as LogGP [38], LogGPC [39], and LogPQ [40], were introduced to improve the model. LogPQ includes communication queues for sending, receiving, and transferring operations. LogGP introduces a new parameter G, defined as the time per byte for a long message (gap per byte). This allows for the modelling of short and long messages. Finally, LogGPC uses LogP parameters for short messages and LogGP parameters for longer messages. Its contribution relies on the inclusion of network contention and network interface direct memory access (DMA).

PLogP [41] and mPlogP [42] have been introduced for multicore architectures. The former includes the overhead (sender and receiver), latency, gap message, and number of nodes. It is suitable for modelling inter-node communication, but lacks a memory access model. The latter is proposed as an extension of PlogP. Unlike PlogP, mPlogP considers multi-grain parallelism (through vector parameters), intra-node communication, and inter-level memory access. The parameters of mPlogP are the overhead (o), which includes the overhead of the sender and receiver, latency (l), gap between messages (g), memory access time (m), and number of cores (P).

For CPU/GPU heterogeneous clusters, the work in [43] presents the mHLogGP model based on the mPlogP, LogGP, and LogP models. It is used to predict the performance of point-to-point and broadcast communications, and the running time of parallel algorithms. The model uses parameters such as overhead, latency, gap per byte, gap between messages, and number of computer nodes. The model also helps to estimate possible bottlenecks.

D. COLLECTIVE COMPUTING MODEL

The collective computing model (CCM) [44] 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, $\mathbf{F}$: group of collective functions f, $\mathbf{TF}$: cost functions for each $f \in \mathbf{F}$, $\mathbf{P}$: group of partition functions p, and $\mathbf{TP}$ cost functions for each $p \in \mathbf{P}$.

E. ROOFLINE MODEL

The Roofline [45] is a throughput-oriented performance model for auto-tuning the performance of multicore computers. It provides information about data movement and

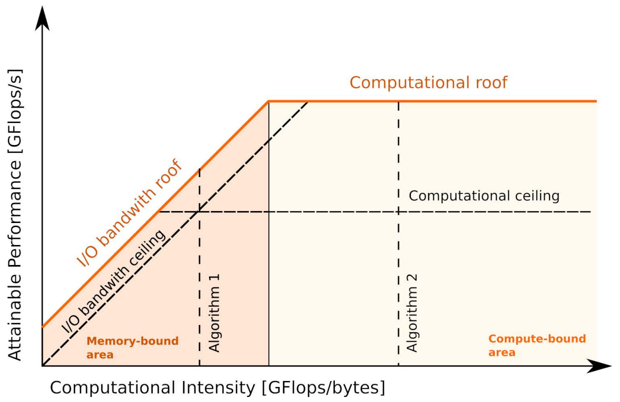

FIGURE 4. Roofline model, based on [48]. The *x*-axis represents the operational or computational intensity (CI) and *y*-axis represents the attainable performance (AP) or throughput. Computational roof and I/O bandwidth roof limit the achievable AP. On the right (yellow area), the algorithms are compute-bound, while on the left (orange area), they are memory-bound.

computation to understand the limitations of the code and combines bandwidth, locality, and different parallelization paradigms. Fig. 4 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. 2, and corresponds to the y-axis [GFlops per second]. Some contributions in the literature, such as [46], [47], extend the Roofline to cache hierarchy (hierarchical Roofline) by considering L1, L2, device memory, and system memory bandwidths.

$$AP[GFLOPS/sec] = \min \begin{cases} PC, \\ PMB \times CI \end{cases} \tag{2}$$

In recent years, this model has been used for performance analysis of different computer architectures and application domains. A toolkit for modelling based on Roofline is presented in [49] for multicore, manycore, and accelerated architectures. Roofline has been applied in the context of deep learning using GPU [50]. The model includes time and complexity to add new features pertinent to applications. The authors in [47] propose a practical methodology for GPU that allows a hierarchical Roofline performance analysis.

F. CLASSIFICATION OF PARALLEL COMPUTING MODELS

Zhang *et al.* [51] classify parallel computing models into three groups based on their evolution over the years and in the memory model of their targeting parallel computers. The first group includes the shared memory parallel computing model (PRAM), which has four approaches: asynchronous, memory contentions, latency-bandwidth, and hierarchical parallelism. The second group includes distributed memory parallel computing models (BSP, CCM, and LogP and its variants). The third group includes hierarchical memory parallel computing models (P-HMM [52], UHM [53], LogP-HMM [54], HPM [55], among others). The authors remark the simplicity, portability, and structured programming style of the BSP model, concluding that BSP offers a better level of abstraction than LogP for designing and programming parallel algorithms.

The third group is based on the speed gap between the processor and the memory system. To reflect the memory access costs, the models incorporate a local memory hierarchy. Models within this last category are subdivided into uniform hierarchical models, LogP extended models, DRAM (h, k) model, and HPM model.

Some models cannot be strictly classified into these three groups. This is the case with the traditional Roofline model [45], which quantifies the traffic between memory and cache rather than between processors and cache. The processor performance depends on the off-chip memory traffic. In contrast, DRAM-only Roofline is extended and improved in the recent hierarchical Roofline [46], [47] supporting different cache levels.

A technical literature survey is presented in [56] for performance modelling and prediction of parallel and distributed computing systems. It analyzes different techniques, mathematical modelling, measurements, and simulations. A recent study by Riahi *et al.* [57] compares analytical, and machine learning models for predicting CPU/GPU data transfer time.

Table 1 presents a comparison of the main features of the models described in this section. 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.

III. FPGA-BASED RECONFIGURABLE HARDWARE ACCELERATORS

FPGA architectures contain a large number of 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, which are 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 been arising.

Integrating a processor and FPGA into a single chip allows the exploitation of different but complementary computational resources of both devices. A performance boost of the system can be achieved by dumping critical functions to the FPGA while maintaining the data transfer quickly and coherently between the devices.

The SoC based on FPGA architecture combines a processing system with programmable logic (FPGA).

TABLE 1. Features of the computing models PRAM, BSP, LogP, CCM, multi-BSP, and Roofline.

Model	Communication			Synchronization Cost	Asynchronous Communication Cost	Computation Cost	Memory Cost	Parameters
	Shared	Distributed	Hierarchical					
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

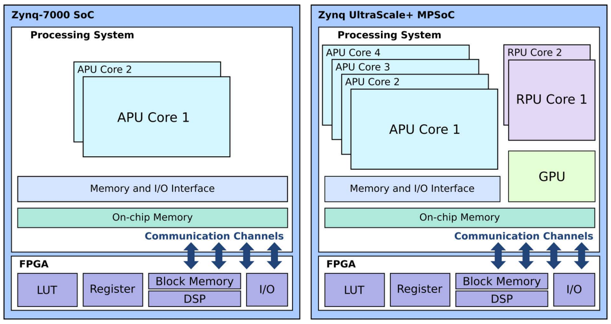

FIGURE 5. Architectures for Zynq-7000 SoC and Zynq UltraScale + MPSoC devices.

The architecture also includes specific interfaces that provide high bandwidth and low latency in the connections between the two parts of the SoC based on FPGA device. The processing system has a fixed architecture formed of a ''hard'' processor and a RAM memory, while the FPGA is completely flexible for hardware design.

Within this context, a processing element (PE) can perform an entire computation containing all the elements required for its replication, which improves the performance of the entire system through coarse-grain parallelism. As an example of this architecture, Fig. 5 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 main 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.

A. HARDWARE/SOFTWARE CO-DESIGN

Hardware/software co-design aims to exploit the inherent features of different technologies, deciding which part of the algorithm should be implemented with sequential instructions (in the processor) and which part in the hardware (such as ASIC or FPGA). Usually, a profiling of the algorithm helps to determine which part is suitable to accelerate. Typically, the most expensive section of the code, in terms of runtime, is a good candidate for hardware acceleration. Regarding communication overhead, its complexity should be minimized between both technologies (that is, between the processor and the FPGA). Also, energy efficiency could be achieved through this technique. Recent contributions in the literature expose the benefits of co-design hardware/software strategy, such as [58], [59], [60], [61], [62].

B. DESIGN SPACE EXPLORATION AND METRICS

HLS tools are used to create RTL components from a high-level of 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 important 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), which are dedicated to optimizing objective functions in the presence of conflicting metrics. In this scenario, trade-off solutions contribute to forming 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 $q \in D$. 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. In the case of area (A) and latency (L) as the objective functions; any hardware design d_i is considered a Pareto-optimal design, and in consequence $d_i \in PF$, 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 [14], as shown in Eq. 3.

$$A(d_i) \leq A(d_n) \;\; and \;\; L(d_i) \leq L(d_n) \tag{3}$$

A survey on MOOA for HLS, presented by Fernandez de Bulnes *et al.* [63], remarks on the expansion

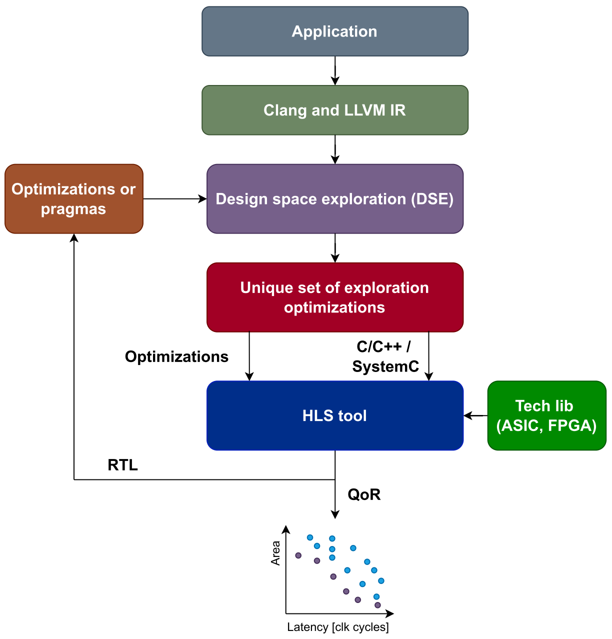

FIGURE 6. Typical DSE framework with HLS in the loop, based on [14].

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. They claim that 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 [64], [65], [66], [67], [68], [69], [70], [71], [72], and [73].

An overview of the general DSE process using HLS tools in the loop, based on [14], is shown in Fig. 6. 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) [74] is obtained from the input code through the Clang front-end compiler [75], 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.

Among the main objective functions associated with FPGA/SoC, we can identify the performance, area, and power. The performance includes the latency (L) and throughput (T). This is directly related to the maximum frequency (f_{max}) of the synthesized design given by $T = f_{max}/L$. The area includes hardware resources: reconfigurable hardware (LUTs, CLBs, and slices) and static hardware (DSPs and BRAMs). The power is the total power consumed (static and dynamic).

Other metrics could be added, such as scalability measured as the number of PE inside the FPGA, bytes per operation, processing system features, and off-chip and on-chip memory bandwidths.

C. TECHNIQUES TO IMPROVE LATENCY, AREA, AND POWER

Different techniques can be used to improve the performance of algorithms running on FPGAs though HLS tools [76]. 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 [6], [8]. Some of the most used optimization techniques are:

- Pipelining: in the presence of sequential operations executed multiple times, this technique allows the insertion of registers at the output of each stage, so that each operation can run in parallel on different input data, increasing the overall throughput at expenses of area. Pipelining can be applied at instruction and function levels.
- Loop unrolling: let us denote f as the unroll factor. For a rolled loop, one iteration is executed at n clock cycles. 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 terms of 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.

Nevertheless, memory performance could be affected by array partition techniques because an improper partitioning leads to generate a large amount of multiplexers, incurring in additional delays [77].

Code restructuring techniques [78], [79], [80], [81], [82] are also used to improve the hardware design of the algorithms. Ferreira *et al.* [83] introduce an approach for automatic code restructuring targeting HLS tools. A detailed survey is presented in [82], 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 to represent 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 [84], [85], [86], [87].

The dynamic power consumption P_d depends on the design and can be improved by considering each element present in the Eq. 4 [88]. As can be noticed, P_d is directly proportional to the clock frequency f, which increases with the square of the power supply V, and it is also affected by the effective capacitance C_i, resource utilization R_i, and switching activity S_i for a given resource i. Hence,

$$P_d = V^2 \times f \times \sum_i C_i \times R_i \times S_i \tag{4}$$

A survey on this topic is presented in [89], considering ultra-low-power techniques for FPGA-based IoT systems. Contributions devoted to improving power consumption on FPGA are presented in [90] and [91].

IV. MODELS, METHODOLOGIES, AND FRAMEWORKS FPGA/SoC

We present the models, methodologies, and frameworks that have been proposed to estimate the performance metrics associated with FPGA/SoC to reduce design times and improve productivity. Some of these models, methodologies, and frameworks propose an exploration of the design space to grant a hardware design with good compromises between different metrics. Other ones include power consumption estimation because low power is one of the main highlights of FPGA-based hardware accelerators.

In this section, we classify models, methodologies, and frameworks into the following categories according to their main features: metrics estimation, FPGA-based DSE, and power consumption estimation.

A. METRICS ESTIMATION

1) METHODOLOGIES

Among the methodologies, we can find the works presented in [92] and [93]. HLScope [92] consists of 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 provides execution times and analyzes various stall causes: external DRAM access, synchronization, and dependency. HLScope+ [93] extends 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 modelling.

Kapre *et al.* [94] present a communication discipline inspired by synchronous dataflow [95] 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.

2) MODELS

In the early stage of the design, models have been applied to FPGA/SoC to mainly estimate latency and area.

Hora *et al.* [96] proposes pipelining circuit RAM (PCRAM), which is a computational model that considers only synchronous circuits. Several algorithms are described, and the model is used to obtain time complexities, leaving for future work the contrast with the experimental results. 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.

A cost model for FPGA partial reconfiguration, proposed by Papadimitriou *et al.* [97], considers all physical elements involved in the reconfiguration process, where each phase contributes to the total reconfiguration time. The authors also explore the parameters that affect the reconfiguration performance.

FlexCL, introduced by Wang *et al.* [98], is an analytical performance model that uses the OpenCL kernel as the input and supplies the performance estimated for the FPGA. A high-level scheme of this model is presented in Fig. 7. The input source code is transformed into an LLVM IR trace through Clang. Information such as the code structure and operation latency is extracted using a kernel analyzer and sent to different models: a computation, communication, and global memory model. As a result of the integration of these three models, the execution time for a given kernel is estimated. FlexCL contributes to identifying performance bottlenecks on FPGA, where PEs, computation units, and kernels have their own models. FlexCL considers eight global memory access patterns; and can also be used to explore the design space to identify solutions under given user constraints.

Currently, Roofline is used for the recognition of the highest performance and potential bottlenecks in FPGA, due to its intuitiveness and simplicity while providing insights about the arithmetic computation and attainable performance. An extended version of the Roofline multicore model for hardware accelerators is presented by Silva *et al.* [48], 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 include the scalability parameter to determine the PE replication factor, considering the available resources and resource utilization per PE. Starting from this initial proposal, contributions in the literature [99], [100] extend this model to FPGA devices. Calore *et al.* [99] present 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.* [100] extend the empirical Roofline toolkit (ERT) to FPGAs, presenting a benchmark for the energy efficiency.

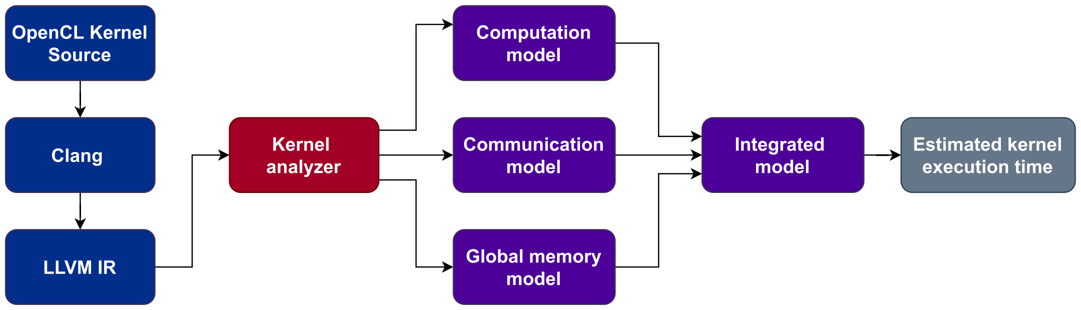

FIGURE 7. High-level overview of FlexCL, based on [98]. The input is the OpenCL kernel code, which is transformed to LLVM IR through Clang. Information from the source code is extracted by a kernel analyzer, which is sent to a computation model, a communication model, and a global memory model. The results of each model are integrated in one model to estimate the final kernel execution time.

3) FRAMEWORKS

Pyramid, developed by Makrani *et al.* [101], 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 with HLS. It is 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 [102], which is an automated hardware optimization tool based on a heuristic model, is used to obtain a good throughput and throughput-to-area ratio for the RTL code generated by HLS.

Wang *et al.* [103] present a framework based on a performance analysis model combined with code tuning techniques for OpenCL applications only on FPGAs, assuming that an incremental development model is adopted by designers [104]. The model includes four FPGA-centric metrics to detect possible bottlenecks related to memory, parallelism, and computation.

4) SUMMARY

For metric estimation, a few contributions have considered the use of the traditional parallel computing models such as BSP and PRAM [94], [96] on FPGA. Nevertheless, the adoption of the Roofline model for estimating performance and bottlenecks on FPGA devices has been widely adopted due to its intuitiveness and simplicity [48], [99], [100].

Furthermore, the differences between the metric estimation reported by HLS tools and the post-implementation results are a key point to consider when designing the estimators of performance metrics [101].

B. FPGA-BASED 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 methodologies, models, and frameworks have been proposed based on the analysis of HLS directives, where the exploration of the design space [105], [106] is important because it increases exponentially with the use of directives. The challenge is to find a set of hardware

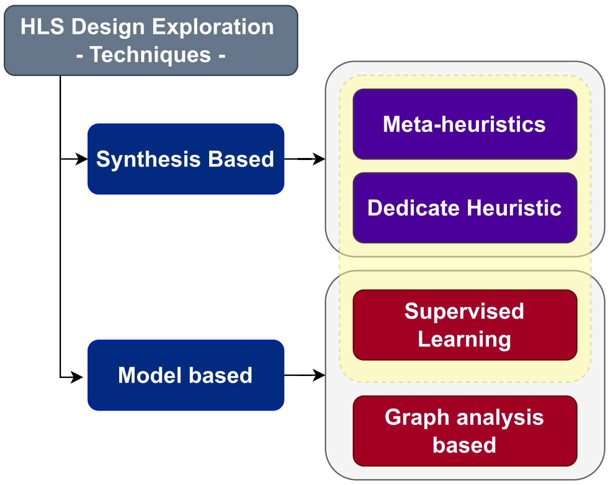

FIGURE 8. Classification of HLS DSE techniques, based on [14].

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.

The comparison among diverse design space explorers is useful for observing the strengths and weaknesses of each. This can be achieved using benchmarks, composed of computational kernels suitable for hardware acceleration. Some of these are MachSuite [107], CHStone (C-based) [108], S2CBench (SystemC-based) [109], Rosetta [110], and Spector (OpenCL-based) [111].

Surveys related to this topic are presented in [63] and [14]. In particular, the last one proposes a classification of HLS DSE techniques into two groups, as depicted in Fig. 8: synthesis-based and model-based. In this classification, the third category is composed of a combination of supervised learning and DSE synthesis-based techniques.

According to Sohrabizadeh *et al.* [15], HLS DSE can be developed using model-based and model-free techniques. Model-based techniques comprise tools and methodologies that use analytical models. They estimate the resources and performance of each point in the design space. Model-free techniques include approaches in which the HLS tool is treated as a black box, such as Bayesian optimization and reinforcement learning techniques [112], [113], [114], [115].

1) METHODOLOGIES

Roofline model has been introduced within methodologies to explore the design space, targeting HPC applications based on HLS [116], [117], [118].

Nabi *et al.* [117] propose 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 pattern. The Roofline model is the base for the design space explorer and is used to assist the selection of the best instance to be downloaded into the hardware. Additionally, the authors propose an intermediate representation language (TyTra-IR). For the calculation of resource utilization to obtain scalability of the system, the authors consider a maximum utilization of the FPGA of 80%, as suggested by [119].

Siracusa *et al.* [118] propose a DSE methodology, presented in Fig. 9. The system input is the C/C++ source code, which is translated to an LLVM IR trace, obtaining the baseline of performance estimation and resource utilization through the synthesis process. From this base implementation, the Roofline model chart (RooflineOrig) determines memory bottlenecks. Afterward, an automated DSE estimates resources and performance, generating the optimal design points. The Roofline for the best feasible design is plotted along with the RooflineOrig chart, to compare the current design's performance and the performance of the solution derived by the DSE. The explorer includes resource sharing and HLS-specific IR optimizations during sample estimations. This work is extended in [116], with the hierarchical version of Roofline, estimating peak performance analytically and integrating a guide to reaching memory-transfer and data-locality optimizations.

Ferretti *et al.* [120] propose a method for inferring knowledge from past design explorations, as shown in Fig. 10. The authors introduce 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), combining CSD and SE. On the other side, the LCS metric is used to measure the similarity between the actual and previous DSE stored in a database.

COSMOS, an automatic and scalable methodology for DSE, is introduced by Piccolboni *et al.* [121] 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 is 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.

The adaptive threshold non-Pareto elimination strategy (ATNE) [122] focuses on inaccuracy estimation, to address the exploration of the design space on FPGA for implementations based on OpenCL. The ATNE algorithm is based on a random forest for regression. The prediction quality is obtained using two metrics: average distance from reference set (ADRS) and hypervolume error (HVE). The results are shown for matrix multiplication, Sobel filter, finite impulse response filter (FIR), histogram, and discrete cosine transform.

Xu *et al.* [123] propose a methodology for performing DSE using MPSoC devices. This work presents three methods to automatically carry out the exploration: two based on simulation (cycle-accurate and fast cycle-accurate) and one based on hardware acceleration. For this purpose, the authors consider several IP cores in an FPGA. The proposed methodology is called fast explorer for behavioral systems (FEBS), and it accepts the number N of IP cores and their testbenches as input. The output is a set of dominant systems with area vs performance trade-off. In this methodology, design space exploration is performed for each IP core. The general overview for this design space explorer is shown in Fig. 11.

2) MODELS

Lo *et al.* [113] propose 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.* [124] propose the mixed-sharing multidomain model for reusing the knowledge obtained from previous HLS DSE whereas exploring a new target design space, showing its effectiveness when approximating quality of results (QoR) without running HLS tools.

Dai *et al.* [125] present a fast and accurate QoR estimation based on HLS. For this purpose, they use final HLS reports from a set of synthesized applications to identify relevant features and metrics, and construct the dataset to be used for training machine learning models (linear regression, artificial neural networks, and gradient tree boosting). To create the dataset, the authors employ the information obtained from HLS reports for different directives and targeting different FPGA platforms. 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 obtain 234 features, which were reduced to 87 after an elimination process to remove irrelevant features.

Other models focus on the DSE process are presented in [126], [127], [128], and [129].

3) FRAMEWORKS

Mehrabi *et al.* propose Prospector framework [114], which uses Bayesian techniques to obtain the best configurations

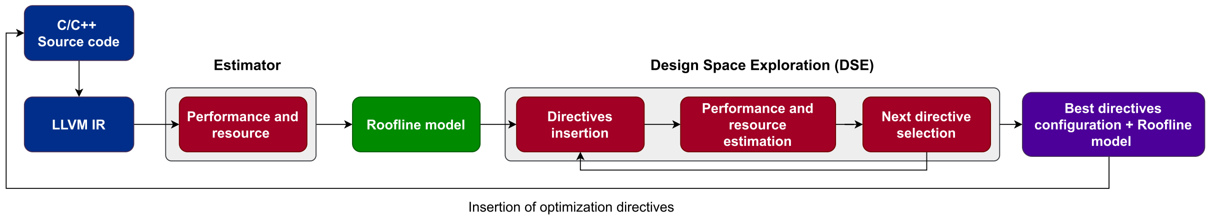
C/C++ Source code
LLVM IR
Estimator
Performance and resource
Roofline model
Design Space Exploration (DSE)
Directives insertion
Performance and resource estimation
Next directive selection
Best directives configuration + Roofline model
Insertion of optimization directives

FIGURE 9. A DSE methodology presented in [116], [118]. 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.

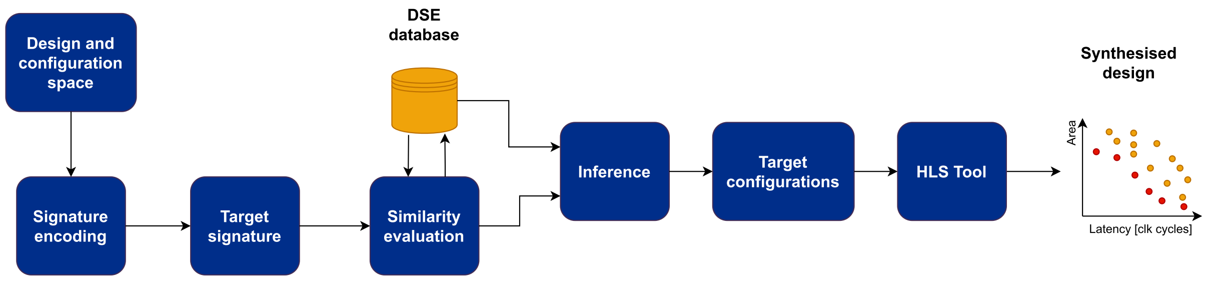

FIGURE 10. A DSE methodology presented in [120] that uses past design explorations to infer knowledge. The signature encoding is used to create a string with the design and configuration spaces. The new signature is compared with the ones obtained from previous DSE (DSE database). After the similarity evaluation, the signature selected is used as input for the inference stage, to finally obtain the optimal configuration.

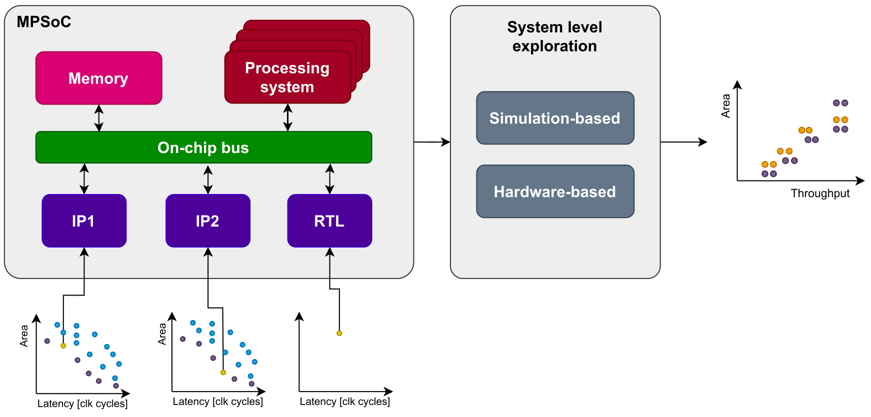

FIGURE 11. MPSoC DSE, based on [123]. Different IP cores coexist in the MPSoC: some developed with HLS tools (IP1 and IP2) and others using RTL description. A design space is generated with the HLS tools. The system level exploration receives as input the number of IP cores described in ANSI-C or SystemC and their testbenches. The output is a Pareto-design with throughput-area trade-off. The system level exploration is composed by three methods: two based on simulation and one based on hardware acceleration.

with fewer resources and reduced latency near Pareto-efficient designs. The HLS tool is considered as a black box (or function), which has to be modelled and optimized. Prospector is shown in Fig. 12, where the inputs are the source code, clock frequency, and directives, and the outputs are the synthesized designs. The Bayesian optimization unit (BOU) is used to explore the design space and control the selection of directives. The HLS tool is used to generate RTL from the high-level source code. At the end of the process, the framework can obtain different designs with a latency-area trade-off, which belong to the Pareto frontier.

Lin-Analyzer [130] is a tool that allows accurate and fast FPGA performance estimation and DSE, considering fine-grained parallelism. With this framework, 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 assume that DSP and BRAM are the bottlenecks in accelerator designs. The communication cost between the FPGA and global memory is not considered. The framework is divided into three main stages: instrumentation, optimization of dynamic data

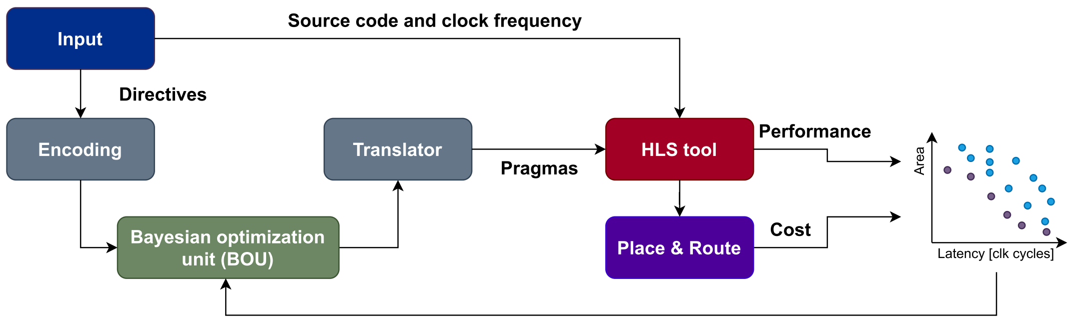

FIGURE 12. **Prospector framework, based on [114]. The inputs are the source code, clock frequency, and directives; and the outputs are the synthesized designs with a trade-off between latency and area. The directives are encoded and sent to the BOU. Source code and clock frequency are the inputs for HLS Tools. Performance and cost values are obtained from HLS tool and Place & Route process.**

dependence graph (DDDG) generation, and DDDG scheduling. In the last stage, latency is used as a performance metric under resource constraints. Lina is proposed in [131] as an extension of Lin-Analyzer, and it includes non-perfect loop nests and timing analyses.

MPSeeker is proposed by Zhong *et al.* [132] to estimate the performance and resource utilization from 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, while Lin-Analyzer is used for BRAM and DSP estimation. The authors also extend the features of Lin-Analyzer 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.* [78] present a DSE and clock cycle estimator using HLS, including code transformations in the presence of variable loop bounds. They propose 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.

COMBA [77], [133] is a framework that focuses on selecting the optimal configuration of directives in HLS, taking into account the use and availability of hardware resources, and provides an estimation of performance and resource utilization. The authors propose the 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 is composed of a recursive data collector, analytical models (latency and resources), and DSE, is presented in Fig. 13. In COMBA, the input is the 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. MGDSE-II then evaluates the configuration and establishes the next set of directives to be applied to the input code. This iteration is repeated until a high-performance configuration is obtained.

Ferretti *et al.* [134] present a framework for HLS DSE using a cluster-based heuristic integrally developed in MATLAB. The algorithm identifies different clusters in the DSE, reducing the number of regions to be analyzed; intra-clustering is performed, followed by inter-cluster exploration. A lattice-traversing DSE framework [135] is proposed to explore the design space by transforming it into a lattice representation. The framework includes three stages: lattice creation and initial sampling, selection of lattice Pareto-neighbours, and synthesis and lattice labelling.

IronMan [115] 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), multi-objective DSE engine based on reinforcement-learning (RLMD), and code transformer (CT). One of the main features of this framework is that it retrieves the final code with the discovered optimizations, ready to generate the corresponding RTL through HLS.

Sherlock [136], introduced by Gautier *et al.*, is a DSE framework based on multi-objective optimizations devoted to find Pareto-optimal solutions (or Pareto front), handling multiple conflicting optimization objectives. This framework uses active learning to exploit a surrogate design space model to find the Pareto-optimal designs as quickly as possible.

Others frameworks devoted to DSE are introduced in [15], [136], [137], and [138].

4) SUMMARY

A summary of most of the contributions devised for DSE and presented in this section are listed in Table 2, considering the following aspects:

- Reference.
- Pruning of the design space (P-DS).
- Whether it is based on the Roofline model.
- Whether it considers quality of results (QoR) in relation to the place and route estimation.
- Whether it applies transfer learning (TL).

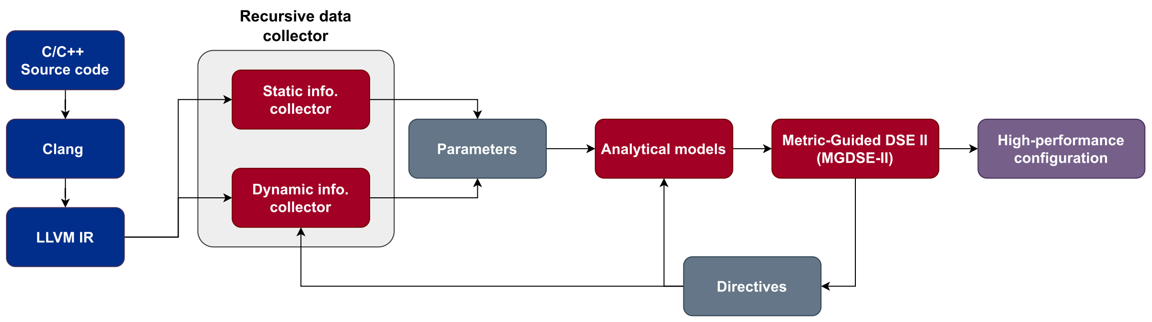

FIGURE 13. COMBA framework overview, based on Zhao *et al.* [77]. LLVM IR is extracted from the source code. This trace is the input for the recursive data collector, which will extract the parameters used by the analytical models (latency and resource). MGDSE-II evaluates the configuration and defines the next set of directives to be applied. The output of the complete flow is the high-performance configuration.

TABLE 2. Summary of most of the contributions devised for DSE and presented in this section. The acronyms used in the table are: P-DS: pruning of the design space, QoR: quality of results in relation to the place and route estimation, TL: transfer learning, N Resource: number of estimated resources or NS (not specified).

Paper	P-DS	Roofline	QoR	TL	N Resource
[133]	-	-	-	-	2
[140]	x	-	-	x	2
[116]	x	-	-	x	1
[135]	-	-	-	-	4
[124]	x	-	-	x	NS
[137]	x	-	-	x	NS
[128]	x	-	x	-	4
[80]	-	-	-	-	2
[120]	-	x	-	-	4
[127]	-	-	-	x	NS
[79], [136]	-	-	-	-	3
[126]	-	-	-	-	NS
[123]	x	-	-	x	NS
[121]	-	x	-	-	4
[16]	x	-	-	x	NS
[119]	-	x	-	-	4
[139]	x	-	-	-	4

- The amount of estimated resources (N Resource): 1 stands for one resource, 2 for two, and so on. NS stands for not specified.

From Table 2, only a few contributions include more than two aspects when developing DSE. A design space explorer can benefit from a reduction of the design space by focusing on obtaining design points near the Pareto frontier, a parallel computing model to guide performance estimation, a good 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.

There are contributions that only estimate some FPGA resources, as follows. LUT-latency trade-off is estimated by [113], BRAM and LUT are computed by [137]. COMBA [77], [133] estimates DSP, BRAM, and LUT. Lin-Analyzer [130] computes BRAM and DSP, whereas MPSeeker [132] 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 [125]. A challenge with HLS tools is efficiently predicting resource sharing for unrolling factors and array partitions when using HLS pragmas. [78], [118].

C. POWER CONSUMPTION ESTIMATION

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 in the first steps of the design process for a given application.

FPGA vendors have proposed different tools to estimate power consumption, such as Maxim® integrated power solution with a USB-to-PMBus interface dongle [139], USB interface adapter EVM from Texas Instruments® [140], Xilinx® power estimator based on spreadsheets (XPE) [141], and Intel® FPGA power and thermal calculator [142]. With FPGA/SoC devices, power is classified as static (fixed and technology-dependent) and dynamic (data and design-dependent). A recent survey on power consumption in FPGA and ASIC devices [17] classifies the techniques for its estimation into analytical, table-based, polynomial-based, and neural networks.

1) METHODOLOGIES

KAPow, proposed by Davis *et al.* [143], is an online activity-based power methodology that includes a signal pruning strategy. The flow has two phases: signal selection (nets with strong relationships between activity and power) and instrumentation (implying the accumulation of events to monitor the relevant signals). A linear model is used to estimate the power contribution of the overall system by computing the power consumption of each IP core.

In the context of approximate computing, Xu *et al.* [144] investigate the use of linear regression and multilayer perceptron (MLP) models to generate a new approximated RTL design with a trade-off between area and power. Using this approach, the search space is extended by reducing the precision of the weights obtained for the predictive models. The proposed method is divided into three stages: kernel extraction and training data generation, model fitting and substitution, and model precision optimization with bit width reduction.

2) MODELS

Lorandel *et al.* [145] propose the use of neural networks to estimate the dynamic power consumption and output signal activities for different IP cores involved in a system. In this study, two stages are considered: IP characterization and high-level system modelling. Nasser *et al.* [146] present a model for the characterization phase by extracting the relevant information for each component that has an impact on power.

Tripathi *et al.* [147] introduce an MLP architecture to calculate power consumption, using LLVM IR instructions as input, and modelling only dynamic power.

Verma *et al.* [148] present a power estimation model that improves the Deng's model [149], and is designed using nonlinear regression techniques. For this purpose, they use the power data of different types of digital circuits (described in VHDL) after the synthesis process. The data is divided into designs with and without clock gating, and based on this separation, two power models are developed.

In [150] two techniques are proposed by Verma *et al.* remarking the importance of predicting the power consumption in an early stage of the accelerator design: a heuristic approach based on a backpropagation neural network and a regression based on statistics.

FlexCL is extended in [151] through the incorporation of three modes of communication for the memory model: direct, burst, and stream access patterns, and an analytical power model for dynamic and static power.

3) FRAMEWORKS

HLSPredict, developed by O'Neal *et al.* [152], 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 are 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 HLS, the functionality of the IP core is encapsulated and integrated within the hardware template.

HL-Pow, proposed by Lin *et al.* [153], 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 is constructed by a feature construction (HLS report) and power collection (post-implementation report), with a total of 256 elements per feature. The experiments are performed with different machine learning models, including linear regression, support vector machines, tree-based models, and neural networks.

PowerGear, described by Lin *et al.* [154], is a graph-learning-assisted power estimator for FPGA HLS, and 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 the hardware design that affects the power modelling. The authors benefit from the HLS front-end and HLS back-end to recover dataflow graphs because it is possible to obtain the IR traces and finite state machine with data path 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.* [155], estimates the performance, power, and area of accelerators. It generates a dependence graph from the input code and produces a fast cycle estimate before RTL construction.

HAPE, presented by Makni *et al.* [156], 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 present in a source code containing the directives provided by the designer. HAPE integrates Lin-Analyzer for computation cost.

4) SUMMARY

Regarding the power consumption, there is an evident trend in estimating this metric in the early stages of design using HLS tools. Moreover, some of the presented frameworks integrate the performance, power, and area estimations with a DSE engine.

D. SUMMARY AND DISCUSSION

The studies described in this section are summarized in Table 3, including for each one:

- Reference and year of publication.
- Whether it is a model, a methodology, or a framework.
    - In the case of a model, the number of input parameters is included. For example, the model presented in [157] uses more than 10 input parameters (10+), and the model presented in [98] uses 21 parameters. The symbol (−) indicates that the number of parameters is not defined in the corresponding study.
- Whether it includes DSE.

- Estimated metrics: area (A), latency (L), power consumption (P), or other metric such as throughput (T), quality of results (QoR), throughput-reconfiguration time (T-RT), energy (E), speed-up (S), or Communication (C).
- Programming language of the input code: SystemC (S-C), C/C++, Impulse-C (I-C), HDL, or OpenCL.
- Optimized designs, divided into Pareto-designs (Pareto) and high performance configuration (HP Config).
- Techniques used to implement the proposed approaches: statistical, analytical, machine learning (ML), and others.

Table 3 shows that the described contributions are fairly distributed between models (35%) and frameworks (41%), whereas 24% propose methodologies. In line with the growing tendency in developing design space explorers, 55.2% of the contributions include DSE.

We can observe 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: 65.3%, 57%, and 26.5%, respectively. We also present this result in Fig. 14. The area and latency metrics are widely estimated because reconfigurable platforms are resource constrained and are used for algorithm acceleration.

Concerning the 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 3 also shows that the C/C++ source code is preferably used as input (65.3%), and the Pareto frontier is the most applied solution to obtain optimal designs (33%) 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 (43% and 41%, respectively), as shown in Fig. 15. However, in the last years, machine learning is the most widely used technique.

The models, methodologies, and frameworks for metric estimation, FPGA-based DSE, and power consumption described in this section are illustrated in Fig. 16. 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.

Fig. 17 summarizes the main topics presented in the research works reviewed in this paper and discussed in this section.

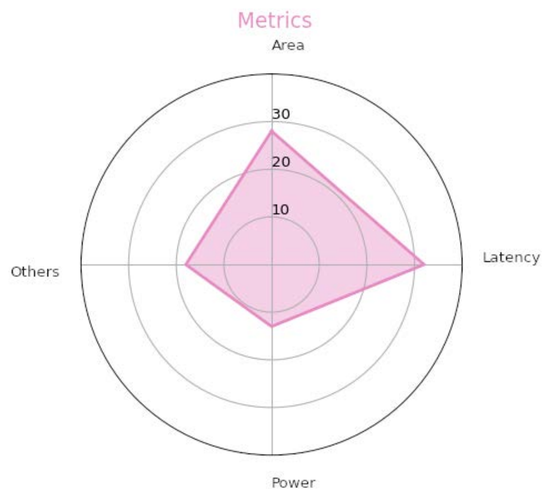

FIGURE 14. Radar plot for metrics used by models, methodologies, and frameworks for FPGA-based hardware accelerators.

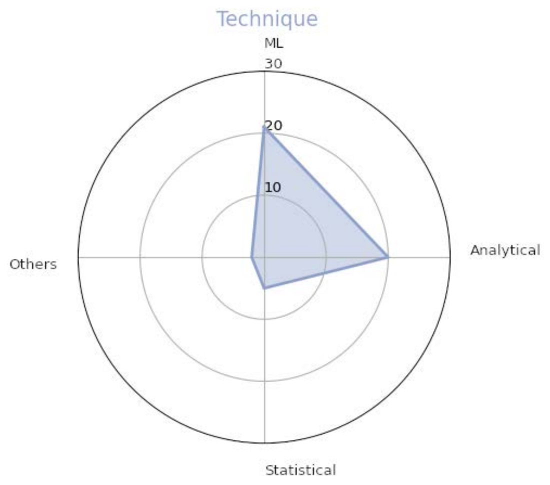

FIGURE 15. Radar plot for the techniques used by models, methodologies, and frameworks for FPGA-based hardware accelerators.

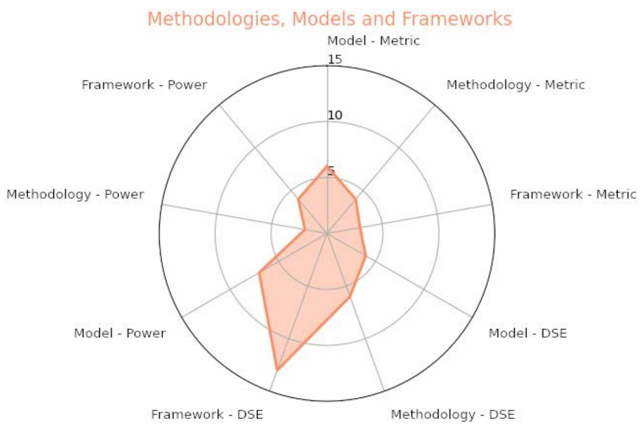

FIGURE 16. Radar plot for models, methodologies, and frameworks for metric estimation, FPGA-based DSE, and power consumption.

V. INTEGRATION IN DIFFERENT RESEARCH FIELDS

In this section, we present contributions in the literature that propose models and frameworks for specific hardware

TABLE 3. Contributions presented in the literature for metric estimation, FPGA-based DSE, and power consumption. 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.

						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
[49]	2013	x (10+)							C		x							x		
[158]	2014			Aladdin			x	x			x						x			
[133]	2016			Lin-Analyzer	x	x	x				x							x		
[140]	2016			x	x	x	x				x					x		x	x	
[106]	2016			x					C					x				x		
[125]	2016		x		x	x	x							x	x				x	
[116]	2016	x (-)			x	x	x				x					x	x			
[148]	2016	x (-)						x					x			x	x			
[79]	2017			COMBA	x	x	x				x					x		x		
[120]	2017		TyTra		x	x	x		C					x						
[135]	2017			MPSeeker	x	x	x		C		x							x	x	
[95]	2017		HLScope			x	x		C		x							x		
[96]	2017		HLScope+				x		C		x							x		
[101]	2017	FlexCL (10+)				x	x		C	x				x				x		
[124]	2017		COSMOS		x	x	x			x					x					PN
[80]	2018			x	x	x	x				x					x		x		
[149]	2018	x (-)						x				x						x		
[159]	2018			HAPE		x		x			x				x			x		
[154]	2018	FlexCL (21)				x	x	x	C					x				x		
[146]	2018		KAPow					x						x					x	
[138]	2018			x	x	x	x				x				x				x	
[137]	2018			x	x	x	x				x				x				x	
[99]	2018	x (-)					x						x					x		
[128]	2018	x (10+)					x		QoR	x	x								x	
[155]	2018			HLSPredict			x			x	x								x	
[134]	2019			Lina	x	x	x				x							x		
[104]	2019			Pyramid		x	x		T	x									x	
[129]	2019	x (-)			x	x	x			x	x				x					
[141]	2019			XPPE	x		x		S		x								x	
[147]	2019		x		x	x		x			x				x				x	
[151]	2019	x (10+)						x	E				x					x		
[153]	2019	x (11)						x	E				x				x		x	
[126]	2019		x												x					
[127]	2020	x (-)			x	x	x		QoR					x	x				x	
[130]	2020	x (10+)			x				C		x								x	
[123]	2020		x		x	x	x				x				x				x	
[150]	2020	x (10+)			x			x			x								x	
[156]	2020			HL-Pow				x			x				x				x	
[121]	2020		x		x	x	x		T, C		x				x			x		
[103]	2021	x(-)			x				T, C		x				x			x		
[119]	2021		x		x	x	x		T, C		x				x			x		
[102]	2021	x(-)				x	x		T, C		x				x				x	
[117]	2021			Prospector	x	x	x				x				x		x			
[118]	2021			IronMan	x	x	x				x					x			x	
[16]	2021			AutoDSE	x	x	x				x					x			x	MH
[157]	2022			PowerGear	x			x			x				x				x	
[139]	2022			Sherlock	x						x				x				x	
Percentage		35%	24%	41%	55.2%	57%	65.3%	26.5%	32%	13%	65.3%	2%	8%	14.3%	33%	14.3%	10%	41%	43%	4%

acceleration applications. Some of them are based on general models such as Roofline. We show that the frameworks and models for FPGA/SoC are used in diverse research areas, exposing their benefits in the design of hardware.

A. MODELS

The Roofline model has been introduced to assist the designer when targeting hardware acceleration of HPC applications, so as to explore the design space, estimate the performance,

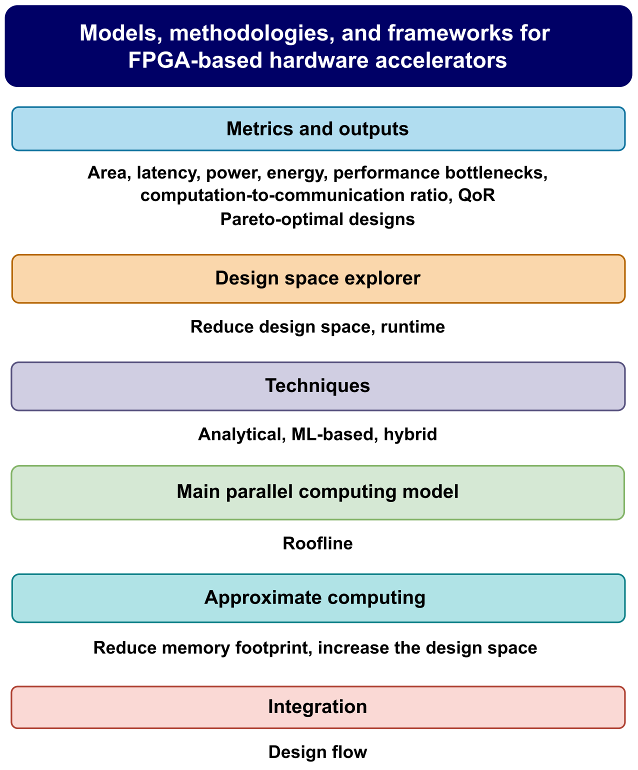
Models, methodologies, and frameworks for FPGA-based hardware accelerators
Metrics and outputs
Area, latency, power, energy, performance bottlenecks, computation-to-communication ratio, QoR
Pareto-optimal designs
Design space explorer
Reduce design space, runtime
Techniques
Analytical, ML-based, hybrid
Main parallel computing model
Roofline
Approximate computing
Reduce memory footprint, increase the design space
Integration
Design flow

FIGURE 17. General summary of the surveyed contributions presented in this section.

and evaluate the throughput due to its dependency on communication and computation.

Roofline is applied by Du *et al.* [158] in the acceleration of the stencil computation kernels, by Karp *et al.* [159] for the hardware implementation of a spectral element method, and by Nagasu *et al.* [160] in the context of an FPGA-based tsunami simulation.

In computational fluid dynamics (CFD), Du *et al.* [161] present an FPGA-based CFD simulation architecture using a performance model to guide the DSE while achieving the maximum performance of the lattice Boltzmann method, searching for an optimal combination of the parameters of the unroll directive.

Reggiani *et al.* [162] present the acceleration of iterative stencil computation using Verilog to describe hardware. An analytical model that considers memory transfer and computation is proposed to estimate the attainable performance of the accelerator and speedup the DSE.

Through efficiency degradation, it is possible to obtain hardware designs with higher performance, lower power consumption, and lower resource utilization at the cost of QoR. Manuel *et al.* [129] propose a DSE in the context of model-based approximate computing for image processing using a multi-objective genetic algorithm, finding 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 [163], [164], [165]. Because of this trend, models for FPGA-based architectures are being developed to accelerate ML applications with efficient exploitation of hardware resources, with the aim of improving productivity in the design phase [166], [167], [168].

Resource and performance models are proposed by Reggiani *et al.* [169] for convolutional neural network (CNN) accelerators, to drive an automatic Pareto-optimal DSE, exploring network performance on different hardware platforms. These models are applied to convolutional cores, which are critical components of the design, directly affecting 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.* [170] present an analytical model for deep CNN design, which is useful for obtaining the computational cost and inferring the required memory bandwidth for the hardware design.

CaFPGA, developed by Xu *et al.* [171], is an FPGA-based DSE for CNN that focuses on convolutional and fully connected layers. To improve the productivity in the design phase, the authors propose an automatic generation model, including incremental searching and flexible layer-folding algorithms, considering that the on-chip memory is a limited resource in FPGA. The analysis of the design space is performed using time, resource, memory, and performance models.

Shan *et al.* introduce [172] a CNN multi-kernel application and its implementation on AWS-F1, where an analytical model is 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 is employed as a performance predictor for FPGA-based CNN accelerators [173], [174], [175], [176]. Ayat *et al.* [173] present an optimization for an FPGA-based CNN accelerator for energy efficiency. Xie *et al.* [174] use this model to quantitatively analyze the design phase of a CNN accelerator, depending on the available computing and memory resources. Park *et al.* [175] propose a model based on Roofline to effectively compute convolutional layers using metrics such as throughput, on-chip memory, off-chip memory bandwidth, and the computation-to-communication ratio.

Ma *et al.* [176] introduce a coarse-grained analytical performance model for CNN accelerators. For this purpose, the modelling of DRAM access, latency, and on-chip buffer is analyzed to obtain the final model. Regarding DSE, convolution throughput is the main focus, considering factors such as operating frequency, external memory bandwidth, and loop unrolling variables, using Roofline to analyze the throughput of the CNN accelerator. Resource costs are obtained by considering the knobs loop unrolling and tiling.

Table 4 summarizes the models 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 and fifth columns are the aim and type of model used, respectively, and the last one 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 to improve the throughput of an accelerator, specially when developing hardware for highly demanding applications.

B. FRAMEWORKS

Frameworks (or toolflows) have been proposed to map ML inference and training into SoC-based, integrating models to mainly estimate hardware resource utilization, latency, and throughput. An exhaustive survey is presented in [166].

Concerning training acceleration, Geng *et al.* [177] developed FPDeep, a toolflow for a 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 is used to evaluate the throughput, because of its dependency on communication and computation.

F-CNN, introduced by Zhao *et al.* [178], is an automatic framework for CNN training based on the reconfiguration of a streaming data path at runtime. The proposed models for resource and bandwidth estimation guide the space exploration under design constraints to obtain an optimal performance.

HP-GNN, proposed by Lin *et al.* [179], 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.* [180] present 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 is implemented using manual directive tuning, reinforcement learning, and the hill-climbing methods.

Venieris *et al.* [181] propose 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 latency, throughput, or maximum throughput with a latency constraint. Performance estimation and resource utilization models are proposed for DSE.

Cloud-DNN [182], introduced by Chen *et al.*, is a framework for mapping DNN to cloud-FPGA, generating the corresponding HLS project to obtain the final IP core. The proposed accelerator model is based on hardware resource cost (considering DSP and BRAM) and a performance model for each layer (convolutional, max pooling, and fully connected). A greedy algorithm is employed to search for the best accelerator configuration under constraints such as the DSP, BRAM, bandwidth, and DNN layers.

FRED [183], 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.* present [184] a collaborative framework to obtain OpenCL-based hardware designs for CNN implementation. A DSE based on LoopTrees is generated and pruned to reduce the design space. Fine-grained and coarse-grained analytical models are introduced to generate the final optimized solution. The former estimates the 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.* [185], is a framework that allows the generation of image processing accelerators. Several steps are 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 named Spark-to-FPGA-Accelerator (S2FA), introduced by Yu *et al.* [186], transforms Scala computational kernels based on Apache Spark applications into optimized accelerator designs. For this, a learning-based DSE is 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 is executed in the loop to verify each optimization.

AutoDNNchip [187] is 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 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, while 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 is generated and executed to validate the results.

Table 5 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

TABLE 4. Models used for FPGA/SoC on different research areas.

Paper	Year	Field	Aim of the model	Type of model	Platform
[173]	2016	ML accelerators (CNN)	DSE	Analytical + Roofline	Xilinx Virtex-7
[163]	2017	Tsunami simulations	Scalability and performance	Roofline	Intel Arria 10
[174]	2018	ML accelerators (CNN)	DSE	Analytical	Xilinx Virtex-7
[177]	2018	ML accelerators (CNN)	Performance	Analytical	Xilinx XCVU9P
[165]	2018	Various	Performance	Analytical	Xilinx VC707
[176]	2018	ML accelerators (CNN)	Performance	Roofline	Xilinx Zynq
[172]	2019	ML accelerators (CNN)	Network Perf.	Analytical	Xilinx Virtex-7
[178]	2020	ML accelerators (CNN)	DSE	Roofline	Xilinx Virtex-7
[179]	2020	ML accelerators (CNN)	Performance + DSE	Analytical + Roofline	Intel Arria 10 and Stratix 10
[132]	2020	Image processing	DSE	Analytical	Intel Arria 10
[164]	2020	Fluid dynamics	Performance	Deterministic	Xilinx Alveo
[161]	2020	Fluid dynamics	Performance	Roofline	Xilinx Alveo
[175]	2020	ML accelerators (CNN)	Data transfer and computation	Analytical	Amazon EC2 F1
[162]	2021	Fluid dynamics	Performance	Roofline	Intel Stratix 10 GX2800

research area in which the model is applied. The fourth is the name of the framework and the last is the target platform.

As we can observe, 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.

VI. 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, the structured study we have presented in the previous sections indicates the necessity to address some challenges. Coping with them will permit a more widespread adoption of models, methodologies, and frameworks for performance estimation of HLS-based hardware designs for FPGA/SoC technology.

Even using HLS tools, reconfiguring an FPGA/SoC with an efficient hardware design is a challenging task. This is easily made apparent by some observations:

- 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 skilfully, considering techniques to improve the latency, area, and power, as introduced in Section III-C.
- Code restructuring techniques aid creating efficient FPGA implementations using HLS tools, modifying the original source code of the application according to the FPGA architecture. Suggestions for this topic are presented in [82].
- The number of PE replicas 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 so as to increase the scalability of the system.
- There is a trade-off between the different metrics to be optimized, as was presented in Section III-B. As an 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, throughput, among others.
- The hardware generated through HLS tools is directly associated with the applied directives, but sometimes applying and tuning directives require a 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.

We can cope with the above considerations through models, methodologies, and frameworks to reduce design time, as follows:

- The level of coarse-grain parallelism can be obtained by means of a model such as Roofline, identifying the computation-to-communication ratio, exposing the relationship between communication bottlenecks, computations, and number of replicas, as was presented in Section II-E and demonstrated in contributions such as [48], [118].

TABLE 5. Utilization of frameworks FPGA/SoC on different research areas. PDR: Partial dynamic reconfiguration.

Paper	Year	Field	Framework	Platform
[186]	2016	Real-time applications with PDR	FRED	Xilinx Zynq 7010
[188]	2017	Image processing	Hipacc	Xilinx Kintex 7 and Intel Stratix V
[181]	2017	ML accelerators (training)	F-CNN	Intel Stratix V
[184]	2017	ML accelerators (inference)	fpgaConvNet	Xilinx Zynq 7020/45
[189]	2018	Big data analytics	S2FA	Amazon EC2 F1
[187]	2018	ML accelerators	Collaborative framework	Xilinx Virtex 7
[180]	2019	ML accelerators (training)	FPDeep	Xilinx VC709 (x8)
[185]	2019	ML accelerators (inference)	Cloud-DNN	Amazon EC2 F1 and VU9P
[183]	2020	ML accelerators (inference)	CNN2Gate	Intel Cyclone V and Arria 10
[190]	2020	DNN chip design (inference)	AutoDNNchip	Xilinx Ultra96 and ASIC
[182]	2022	ML accelerators (training)	HP-GNN	CPU + Xilinx Alveo U250

- Design space explorers aim to identify the optimal combination of directives to obtain an HLS-based hardware design with the best 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, as was described in Section IV-B.
- Models integrated within a methodology or framework can automatically estimate the performance of HLS-based hardware designs without executing HLS tools, as presented in Section IV.
- Some frameworks and methodologies including DSE provide automatic directive-insertion optimizations and code transformation insights, as in contributions such as [115], [116], [118].

Nevertheless, the literature review shows that a number of challenges has to still be addressed in order to make optimal use of 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 an important factor 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. One way to overcome this is to use approaches such as [113], [121], [124], [152], [156], without the need to run HLS in the loop or reduce its invocation.
- 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 [101] 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 [14].
- Methodologies and frameworks are typically linked to a tool [77], [130], [131], [136]. However, most such tools are not easily available or do not have user support. This is a critical point in the adoption of methodologies and frameworks for performance estimation, which makes

difficult to include them in the design flow. This may be solved by making methodologies and frameworks available to the FPGA designer through a repository system, such as contributions in [77] and [136], among others.

- The integration of frameworks into the different steps of the flow for designing IP cores can be affected by the installation of libraries, dependencies, and tools, such as LLVM IR and Clang, needed for the execution of the frameworks. It should be guaranteed to the user a simple way of installation and maintenance in order to facilitate their integration in the design flow. This concern can be addressed by providing a script with dependencies to be installed, an executable file, or a library package.
- For heterogeneous architectures, the hardware-software co-design can be considered by models, methodologies, and frameworks 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 II (for the sequential part) and the contributions discussed in Section IV (for the FPGA part). In addition, a single parallel model, such as Roofline, can be applied to both architectures.

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, as shown in contributions such as [113], [121], [134], [136], [137].
- 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, as introduced in [136], [188], and [134].
- The DSE engine should guarantee a good compromise among the QoR and performance metrics.
- Approximate computing [189] can lead to an expansion of the design space, generating Pareto-optimal designs with a trade-off between area-power-latency estimation and error computation [129], [144]. 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, as was made in [15], [115], [114], [132], and [77], among others.

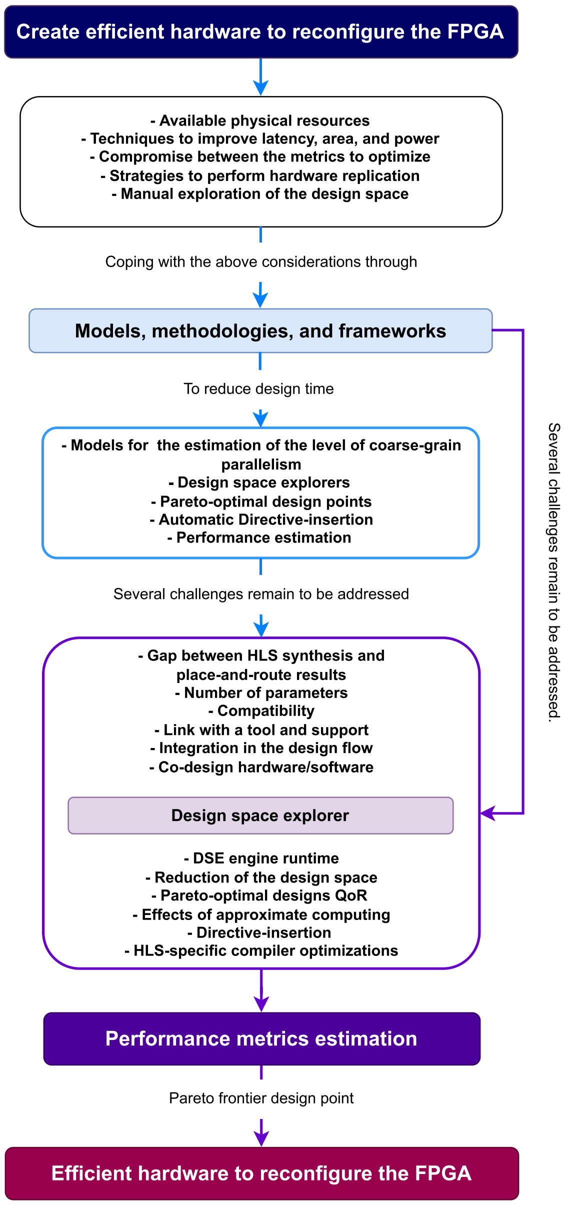
Create efficient hardware to reconfigure the FPGA
- Available physical resources
- Techniques to improve latency, area, and power
- Compromise between the metrics to optimize
- Strategies to perform hardware replication
- Manual exploration of the design space
Coping with the above considerations through
Models, methodologies, and frameworks
To reduce design time
- Models for the estimation of the level of coarse-grain parallelism
- Design space explorers
- Pareto-optimal design points
- Automatic Directive-insertion
- Performance estimation
Several challenges remain to be addressed
- Gap between HLS synthesis and place-and-route results
- Number of parameters
- Compatibility
- Link with a tool and support
- Integration in the design flow
- Co-design hardware/software
Design space explorer
- DSE engine runtime
- Reduction of the design space
- Pareto-optimal designs QoR
- Effects of approximate computing
- Directive-insertion
- HLS-specific compiler optimizations
Several challenges remain to be addressed.
Performance metrics estimation
Pareto frontier design point
Efficient hardware to reconfigure the FPGA

FIGURE 18. Summary of the main aspects presented in this section.

- 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. Contributions in the literature [77], [130], [155] have implemented their own scheduler to obtain solutions with better timing than HLS tools (with no guarantee that HLS will implement it in the same way) [78]. To address this, some contributions [78], [118] use a baseline implementation obtained after HLS synthesis to consider the impact of the compiler optimizations and use the estimated critical path that affects the latency. This implementation is considered the starting point for the DSE engine to search for Pareto-optimal designs. Moreover, 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 [190].

Fig. 18 summarizes the main aspects presented in this section, considering those to create efficient hardware to reconfigure the FPGA, how some of these aspects may be coped through models, methodologies, and frameworks, and the challenges that need to be considered to bridge the gap between designers and FPGA-based reconfigurable hardware accelerators.

VII. CONCLUSION

In this survey, different models, methodologies, and frameworks proposed for metrics estimation, FPGA-based design space exploration, and power consumption estimation on FPGA/SoC have been described. The main features and limitations, as well as trade-offs of these approaches, have been presented, and different challenges to be addressed have been identified.

The integration of models and frameworks in different research areas has also been described, indicating a growing tendency to apply them in the field of machine learning accelerators for diverse applications.

Based on our literature review, it can be observed that existing models, methodologies, and frameworks are very difficult to compare against one another. One reason is the lack of standards limiting their evaluation on different hardware and applications, together with the fact that the different approaches do not analyze the same performance metrics.

In addition, it can be affirmed that the inherent hardware reconfigurability of FPGA/SoC affects the complexity of the associated models. Indeed, the models for FPGA/SoC usually have a higher complexity than those commonly used for CPU, GPU, multicore processors, among other architectures.

We believe this survey can help readers understand the benefits of integrating models, methodologies, and frameworks for FPGA-based hardware accelerators into the design flow. Therefore, the FPGA designer can select the approach that best suits the application, hardware architecture, and programming skills.

The literature review shows that several challenges have to still be addressed to make optimal integration of models, methodologies, and frameworks in the design flow. By highlighting these challenges, this survey reveals what has to be considered to bridge the gap between the FPGA designer and hardware accelerators based on FPGA.

APPENDIX A. LIST OF ACRONYMS

A	Area.
ADRS	Average distance from reference set.
AP	Attainable performance.
ASIC	Application specific integrated circuit.
BRAM	Block RAM.
BSP	Bulk synchronous parallel.
CCM	Collective computing model.
CDFG	Control data flow graph.
CFD	Computational fluid dynamics.
CI	Computational intensity.
CLBs	Configurable logic block.
CNN	Convolutional neural network.
CRCW	Concurrent read concurrent write.
CREW	Concurrent read exclusive write.
CUDA	Compute Unified Device Architecture.
D	Design space.
DDDG	Dynamic data dependence graph.
DMA	Direct memory access.
DNN	Deep neural network.
DSE	Design space exploration.
DSP	Digital signal processor.
ERCW	Exclusive read concurrent write.
EREW	Exclusive read exclusive write.
ERT	Empirical Roofline toolkit.
FF	Flip-Flop.
FIR	Finite impulse response filter.
FPGA	Field programmable gate array.
GNN	Graph neural networks.
HDL	High-level design.
HLS	High-level synthesis.
HPC	High-performance computing.
HPM	Hierarchical model for parallel computations.
HVE	Hypervolume error.
I/O	Input/Output.
IoT	Internet of things.
IP	Intellectual property.
IR	Intermediate representation.
L	Latency.
L1	Level-1 cache memory.
L2	Level-2 cache memory.
LLVM IR	Low-level virtual machine intermediate representation.
LUT	LookUp Table.
ML	Machine learning.
MLP	Multi-layer perceptron.
MOOA	Multi-objective optimization algorithms.
MPSoC	Multiprocessor system on chip.
PC	Peak computation.
PE	Processing element.
PF	Pareto-optimal frontier.
PMB	Peak memory bandwidth.
PRAM	Parallel random access machine.
QoR	Quality of results.
RAM	Random access machine.
RTL	Register transfer level.
SIMD	Single Instruction/Multiple Data.
SoC	System on chip.
SPMD	Single program multiple data.

T	Throughput.
UMH	Uniform Memory Hierarchy Model of Computation.

REFERENCES

[1] Z. Wan, B. Yu, T. Yuang Li, J. Tang, Y. Zhu, Y. Wang, A. Raychowdhury, and S. Liu, "A survey of FPGA-based robotic computing," 2020, *arXiv:2009.06034*.

[2] 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," *Comput. Sci. Rev.*, vol. 27, pp. 135–153, Feb. 2018.

[3] 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.

[4] K. Guo, S. Zeng, J. Yu, Y. Wang, and H. Yang, "A survey of FPGA-based neural network inference accelerators," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 12, no. 1, pp. 1–26, Mar. 2019.

[5] 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 Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 35, no. 10, pp. 1591–1604, Oct. 2016.

[6] *Vivado Design Suite User Guide: High-Level Synthesis. UG-902*, Xilinx, San Jose, CA, USA, 2020. Accessed: Feb. 15, 2022. [Online]. Available: https://docs.xilinx.com/v/u/en-US/ug902-vivado-high-level-synthesis

[7] J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K. Vissers, and Z. Zhang, "High-level synthesis for FPGAs: From prototyping to deployment," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 30, no. 4, pp. 473–491, Apr. 2011.

[8] *Intel High Level Synthesis Compiler, Best Practices Guide. UG-20107*, Intel, Santa Clara, CA, USA, 2020. Accessed: Feb. 15, 2022. [Online]. Available: http://www.audentia-gestion.fr/INTEL/PDF/ug-hls-best-practices.pdf

[9] 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 *Proc. Int. Symp. Field Program. Gate Arrays (FPGA)*, 2011, pp. 33–36.

[10] C. Pilato and F. Ferrandi, "Bambu: A modular framework for the high level synthesis of memory-intensive applications," in *Proc. 23rd Int. Conf. Field Program. Log. Appl.*, Sep. 2013, pp. 1–4.

[11] Y. Liang, K. Rupnow, Y. Li, D. Min, M. Do, and D. Chen, "High-level synthesis: Productivity, performance, and software constraints," *Electr. Comput. Eng.*, vol. 2012, Jan. 2012, Art. no. 649057.

[12] *The OpenCL Specification, Version 1.1*, Khronos OpenCL Work. Group, 2011. Accessed: Feb. 15, 2022. [Online]. Available: https://registry.khronos.org/OpenCL/specs/opencl-1.1.pdf

[13] 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, Dec. 2007.

[14] B. C. Schafer and Z. Wang, "High-level synthesis design space exploration: Past, present, and future," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 10, pp. 2628–2639, Oct. 2020.

[15] A. Sohrabizadeh, C. H. Yu, M. Gao, and J. Cong, "AutoDSE: Enabling software programmers to design efficient FPGA accelerators," *ACM Trans. Design Autom. Electron. Syst.*, vol. 27, no. 4, pp. 1–27, Jul. 2021.

[16] M. W. Numan, B. J. Phillips, G. S. Puddy, and K. Falkner, "Towards automatic high-level code deployment on reconfigurable platforms: A survey of high-level synthesis tools and toolchains," *IEEE Access*, vol. 8, pp. 174692–174722, 2020.

[17] Y. Nasser, J. Lorandel, J.-C. Prevotet, and M. Helard, "RTL to transistor level power modeling and estimation techniques for FPGA and ASIC: A survey," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 40, no. 3, pp. 479–493, Mar. 2021.

[18] K. O'Neal and P. Brisk, "Predictive modeling for CPU, GPU, and FPGA performance and power consumption: A survey," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Jul. 2018, pp. 763–768.

[19] L. Liu, J. Zhu, Z. Li, Y. Lu, Y. Deng, J. Han, S. Yin, and S. Wei, "A survey of coarse-grained reconfigurable architecture and design: Taxonomy, challenges, and applications," *ACM Comput. Surveys*, vol. 52, no. 6, pp. 1–39, Nov. 2019.

[20] A. Podobas, K. Sano, and S. Matsuoka, "A survey on coarse-grained reconfigurable architectures from a performance perspective," *IEEE Access*, vol. 8, pp. 146719–146743, 2020.

[21] S. A. Cook and R. A. Reckhow, "Time-bounded random access machines," in *Proc. 4th Annu. ACM Symp. Theory Comput. (STOC)*, 1972, pp. 354–375.

[22] S. Fortune and J. Wyllie, "Parallelism in random access machines," in *Proc. Symp. Theory Comput.*, 1978, pp. 114–118.

[23] P. B. Gibbons, "A more practical PRAM model," in *Proc. 1st Annu. ACM Symp. Parallel Algorithms Architectures (SPAA)*, 1989, pp. 158–168.

[24] L. G. Valiant, "A bridging model for parallel computation," *Commun. ACM*, vol. 33, no. 8, pp. 103–111, Aug. 1990.

[25] L. G. Valiant, "A bridging model for multi-core computing," *J. Comput. Syst. Sci.*, vol. 77, no. 1, pp. 154–166, Jan. 2011.

[26] A. Goldchleger, A. Goldman, U. Hayashida, and F. Kon, "The implementation of the BSP parallel computing model on the InteGrade grid middleware," in *Proc. 3rd Int. Workshop Middleware Grid Comput. (MGC)*, 2005, pp. 1–6.

[27] V. Allombert, F. Gava, and J. Tesson, "Toward performance prediction for multi-BSP programs in ML," in *Proc. Int. Conf. Algorithms Architectures Parallel Process.*, 2018, pp. 159–174.

[28] G. Trabes, V. Gil-Costa, M. Printista, and M. Marin, "Multi-BSP vs. BSP: A case of study for dell AMD multicores," in *Proc. 26th Euromicro Int. Conf. Parallel, Distrib. Netw.-Based Process. (PDP)*, Mar. 2018, pp. 579–587.

[29] A. Savadi, M. Moradi, and H. Deldari, "Multi-DaC programming model: A variant of multi-BSP model for divide-and-conquer algorithms," in *Proc. 7th Workshop Declarative Aspects Appl. Multicore Program. (DAMP)*, 2012, pp. 41–46.

[30] M. Alaniz and S. Nesmachnow, "A semi-automatic approach for parallel problem solving using the multi-BSP model," *Program. Comput. Softw.*, vol. 45, no. 8, pp. 517–531, Dec. 2019.

[31] Z. Zeng and X. Sun, "Electric vehicle regional management system based on the BSP model and multi-information fusion," *Syst. Sci. Control Eng.*, vol. 9, no. sup1, pp. 114–121, Apr. 2021.

[32] 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," *Mach. Learn.*, vol. 110, no. 10, pp. 2867–2903, Oct. 2021.

[33] K. Siddique, Z. Akhtar, H.-G. 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, Sep. 2017.

[34] X. Zhao, M. Papagelis, A. An, B. X. Chen, J. Liu, and Y. Hu, "Elastic bulk synchronous parallel model for distributed deep learning," in *Proc. Int. Conf. Data Mining, (ICDM)*, 2019, pp. 1504–1509.

[35] M. Amaris, D. Cordeiro, A. Goldman, and R. Y. D. Camargo, "A simple BSP-based model to predict execution time in GPU applications," in *Proc. IEEE 22nd Int. Conf. High Perform. Comput. (HiPC)*, Dec. 2015, pp. 285–294.

[36] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken, "LogP: Towards a realistic model of parallel computation," in *Proc. 4th ACM SIGPLAN Symp. Princ. Pract. Parallel Program. (PPOPP)*, 1993, pp. 1–12.

[37] A. Nomura, H. Matsuba, and Y. Ishikawa, "Network performance model for TCP/IP based cluster computing," in *Proc. IEEE Int. Conf. Cluster Comput.*, Sep. 2007, pp. 194–203.

[38] A. Alexandrov, M. F. Ionescu, K. E. Schauser, and C. Scheiman, "LogGP: Incorporating long messages into the LogP model for parallel computation," *J. Parallel Distrib. Comput.*, vol. 44, no. 1, pp. 71–79, Jul. 1997.

[39] C. A. Moritz and M. I. Frank, "LoGPG: Modeling network contention in message-passing programs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 12, no. 4, pp. 404–415, Apr. 2001.

[40] T. Touyama and S. Horiguchi, "Performance evaluation of practical parallel computation model LogPQ," in *Proc. 4th Int. Symp. Parallel Architectures, Algorithms, Netw. (I-SPAN)*, 1999, pp. 216–221.

[41] T. Kielmann, H. E. Bal, and K. Verstoep, "Fast measurement of LogP parameters for message passing platforms," in *Proc. Parallel Distrib. Process. (IPDPS)*, vol. 1800, 2000, pp. 1176–1183.

[42] L. Li, X. Zhang, J. Feng, and X. Dong, "MPlogP: A parallel computation model for heterogeneous multi-core computer," in *Proc. Int. Conf. Cluster, Cloud Grid Comput.*, 2010, pp. 679–684.

[43] G. Liu, Y. Wang, T. Zhao, J. Gu, and D. Li, "MHLogGP: A parallel computation model for CPU/GPU heterogeneous computing cluster," in *Proc. IFIP Int. Conf. Netw. Parallel Comput.*, vol. 7513, 2012, pp. 217–224.

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

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

[46] C. Yang, T. Kurth, and S. Williams, "Hierarchical roofline analysis for GPUs: Accelerating performance optimization for the NERSC-9 perlmutter system," *Concurrency Comput., Pract. Exper.*, vol. 32, no. 20, p. e5547, Oct. 2020.

[47] C. Yang, Y. Wang, T. Kurth, S. Farrell, and S. Williams, "Hierarchical roofline performance analysis for deep learning applications," in *Intelligent Computing* (Lecture Notes in Networks and Systems), vol. 284, K. Arai, Ed. Cham, Switzerland: Springer, 2021, doi: 10.1007/978-3-030-80126-7_35.

[48] 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," *Int. J. Reconfigurable Comput.*, vol. 2013, Jan. 2013, Art. no. 428078.

[49] Y. J. Lo, S. Williams, B. Van Straalen, T. J. Ligocki, M. J. Cordery, N. J. Wright, M. W. Hall, and L. Oliker, "Roofline model toolkit: A practical tool for architectural and program analysis," in *High Perform. Comput. Syst. Perform. Modeling, Benchmarking, Simul.*, 2015, pp. 129–148.

[50] Y. Wang, C. Yang, S. Farrell, Y. Zhang, T. Kurth, and S. Williams, "Time-based roofline for deep learning performance analysis," in *Proc. Workshop Deep Learn. Supercomputers*, Atlanta, GA, USA, Nov. 2020, pp. 10–19.

[51] Y. Zhang, G. Chen, G. Sun, and Q. Miao, "Models of parallel computation: A survey and classification," *Frontiers Comput. Sci. China*, vol. 1, no. 2, pp. 156–165, May 2007.

[52] A. Aggarwal, B. Alpern, A. K. Chandra, and M. Snir, "A model for hierarchical memory," in *Proc. Symp. Theory Comput.*, 1987, pp. 305–314.

[53] B. Alpern, L. Carter, E. Feig, and T. Selker, "The uniform memory hierarchy model of computation," *Algorithmica*, vol. 12, nos. 2–3, pp. 72–109, Sep. 1994.

[54] Z. Li, P. Mills, and J. H. Reif, "Models and resource metrics for parallel and distributed computation," *Parallel Algorithms Appl.*, vol. 8, no. 1, pp. 35–59, 1996.

[55] X. Qiao, S. Chen, and L. T. Yang, "HPM: A hierarchical model for parallel computations," *Int. J. High Perform. Comput. Netw.*, vol. 1, no. 3, pp. 117–127, 2004.

[56] S. Pllana, I. Brandic, and S. Benkner, "Performance modeling and prediction of parallel and distributed computing systems: A survey of the state of the art," in *Proc. 1st Int. Conf. Complex, Intell. Softw. Intensive Syst. (CISIS)*, Apr. 2007, pp. 279–284.

[57] A. Riahi, A. Savadi, and M. Naghibzadeh, "Comparison of analytical and ML-based models for predicting CPU–GPU data transfer time," *Computing*, vol. 102, no. 9, pp. 2099–2116, Sep. 2020.

[58] O. Bringmann, W. Ecker, I. Feldner, A. Frischknecht, C. Gerum, T. Hämäläinen, M. A. Hanif, M. J. Klaiber, D. Mueller-Gritschneder, P. P. Bernardo, S. Prebeck, and M. Shafique, "Automated HW/SW co-design for edge AI: State, challenges and steps ahead: Special session paper," in *Proc. Int. Conf. Hardw./Softw. Codesign Syst. Synthesis (CODES+ ISSS)*, 2021, pp. 11–20, doi: 10.1145/3478684.3479261.

[59] C. Pham-Quoc, X.-Q. Nguyen, and T. N. Thinh, "Towards an FPGA-targeted hardware/software co-design framework for CNN-based edge computing," *Mobile Netw. Appl.*, vol. 174, pp. 1–12, May 2022.

[60] Q. Xiao, S. Zheng, B. Wu, P. Xu, X. Qian, and Y. Liang, "HASCO: Towards agile HArdware and software CO-design for tensor computation," in *Proc. ACM/IEEE 48th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2021, pp. 1055–1068.

[61] Y. Li, R. Chen, B. Sensale-Rodriguez, W. Gao, and C. Yu, "Real-time multi-task diffractive deep neural networks via hardware-software co-design," *Sci. Rep.*, vol. 11, no. 1, pp. 1–9, Dec. 2021.

[62] N. Talati, K. May, A. Behroozi, Y. Yang, K. Kaszyk, C. Vasiladiotis, T. Verma, L. Li, B. Nguyen, J. Sun, and J. M. Morton, "Prodigy: Improving the memory latency of data-indirect irregular workloads using hardware-software co-design," in *Proc. Int. Symp. High-Performance Comput. Archit. (HPCA)*, 2021, pp. 654–667.

[63] D. R. F. de Bulnes, Y. Maldonado, and L. Trujillo, "Development of multiobjective high-level synthesis for FPGAs," *Sci. Program.*, vol. 2020, Jun. 2020, Art. no. 7095048.

[64] Z. Zeng, R. Sedaghat, and A. Sengupta, "A novel framework of optimizing modular computing architecture for multi objective VLSI designs," in *Proc. Int. Conf. Microelectron. (ICM)*, Dec. 2009, pp. 328–331.

[65] 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 Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 12, pp. 2298–2311, Dec. 2018.

[66] D. Roy and A. Sengupta, "Low overhead symmetrical protection of reusable IP core using robust fingerprinting and watermarking during high level synthesis," *Future Gener. Comput. Syst.*, vol. 71, pp. 89–101, Jun. 2017.

[67] L. Piccolboni, P. Mantovani, G. Di Guglielmo, and L. P. Carloni, "Broadening the exploration of the accelerator design space in embedded scalable platforms," in *Proc. IEEE High Perform. Extreme Comput. Conf. (HPEC)*, Sep. 2017, pp. 1–7.

[68] R. Resmi and B. B. T. Sundari, "Allocation of optimal reconfigurable array using graph merging technique," in *Proc. Int. Conf. Embedded Syst. (ICES)*, Jul. 2014, pp. 49–54.

[69] D. S. H. Ram, M. C. Bhuvaneswari, and S. M. Logesh, "A novel evolutionary technique for multi-objective power, area and delay optimization in high level synthesis of datapaths," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI*, Jul. 2011, pp. 290–295.

[70] 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 Evol. Comput.*, vol. 7, pp. 35–46, Dec. 2012.

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

[72] B. C. Schafer and K. Wakabayashi, "Design space exploration acceleration through operation clustering," *IEEE Trans. Comput.-Aided Design Integr.*, vol. 29, no. 1, pp. 153–157, Jan. 2009.

[73] B. C. Schafer, T. Takenaka, and K. Wakabayashi, "Adaptive simulated annealer for high level synthesis design space exploration," in *Proc. Int. Symp. VLSI Design, Autom. Test*, Apr. 2009, pp. 106–109.

[74] C. Lattner and V. Adve, "LLVM: A compilation framework for lifelong program analysis & transformation," in *Proc. Int. Symp. Code Gener. Optim. (CGO)*, 2004, pp. 75–86.

[75] LLVM Developer Group. *Clang*. Accessed: Feb. 1, 2022. [Online]. Available: https://clang.llvm.org

[76] L. Huang, D.-L. Li, K.-P. Wang, T. Gao, and A. Tavares, "A survey on performance optimization of high-level synthesis tools," *J. Comput. Sci. Technol.*, vol. 35, no. 3, pp. 697–720, May 2020.

[77] 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 *Proc. Int. Conf. Comput.-Aided Des. (ICCAD)*, 2017, pp. 430–437, doi: 10.1109/ICCAD.2017.8203809.

[78] Y.-K. Choi and J. Cong, "HLS-based optimization and design space exploration for applications with variable loop bounds," in *Proc. Int. Conf. Comput.-Aided Design (ICCAD)*, 2018, pp. 1–8.

[79] J. S. Monson and B. L. Hutchings, "Using source-level transformations to improve high-level synthesis debug and validation on FPGAs," in *Proc. Int. Symp. Field-Program. Gate Arrays*, 2015, pp. 5–8.

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

[81] R. Campos and J. M. Cardoso, "On data parallelism code restructuring for HLS targeting FPGAs," in *Proc. Int. Parallel Distrib. Process. Symp. Workshops (IPDPSW)*, 2021, pp. 144–151.

[82] 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, doi: 10.1109/TPDS.2020.3039409.

[83] A. C. Ferreira and J. M. Cardoso, "Graph-based code restructuring targeting HLS for FPGAs," in *Proc. Int. Symp. Appl. Reconfigurable Comput.*, 2019, pp. 230–244.

[84] M. Q. Hoang, P. L. Nguyen, H. V. Tran, H. Q. Nguyen, V. T. Nguyen, and C. Vo-Le, "FPGA oriented compression of DNN using layer-targeted weights and activations quantization," in *Proc. IEEE 8th Int. Conf. Commun. Electron. (ICCE)*, Jan. 2021, pp. 157–162.

[85] Q. Zhang, J. Cao, Y. Zhang, S. Zhang, Q. Zhang, and D. Yu, "FPGA implementation of quantized convolutional neural networks," in *Proc. Int. Conf. Commun. Technol. (ICCT)*, 2019, pp. 1605–1610.

[86] P. Bacchus, R. Stewart, and E. Komendantskaya, "Accuracy, training time and hardware efficiency trade-offs for quantized neural networks on FPGAs," in *Proc. Int. Symp. Appl. Reconfigurable Comput.*, 2020, pp. 121–135.

[87] 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," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 14, no. 4, pp. 1–20, Oct. 2018.

[88] N. Grover and M. Soni, "Reduction of power consumption in FPGAs—An overview," *Inf. Eng. Electron. Bus.*, vol. 4, no. 5, p. 50, 2012.

[89] M. Ibro and G. Marinova, "Review on low-power consumption techniques for FPGA-based designs in IoT technology," in *Proc. 16th Int. Conf. Telecommun. (ConTEL)*, Jun. 2021, pp. 110–114.

[90] B. Khaleghi, S. Salamat, M. Imani, and T. Rosing, "FPGA energy efficiency by leveraging thermal margin," in *Proc. IEEE 37th Int. Conf. Comput. Design (ICCD)*, Nov. 2019, pp. 376–384.

[91] H. Kim and K. Choi, "Low power FPGA-SoC design techniques for CNN-based object detection accelerator," in *Proc. IEEE 10th Annu. Ubiquitous Comput., Electron. Mobile Commun. Conf. (UEMCON)*, Oct. 2019, pp. 1130–1134.

[92] Y. Choi and J. Cong, "HLScope: High-level performance debugging for FPGA designs," in *Proc. Annu. Int. Symp. Field-Programmable Custom Comput. Mach. (FCCM)*, 2017, pp. 125–128.

[93] Y. Choi, P. Zhang, P. Li, and J. Cong, "HLScope+: Fast and accurate performance estimation for FPGA HLS," in *Proc. Int. Conf. Comput.-Aided Des. (ICCAD)*, 2017, pp. 691–698.

[94] N. Kapre and H. Patel, "Applying models of computation to OpenCL pipes for FPGA computing," in *Int. Workshop OpenC (IWOCL)*, 2017, pp. 1–9.

[95] E. A. Lee and D. G. Messerschmitt, "Synchronous data flow," *Proc. IEEE*, vol. 75, no. 9, pp. 1235–1245, Sep. 1987.

[96] M. Hora, V. Končický, and J. Tětek, "Theoretical model of computation and algorithms for FPGA-based hardware accelerators," 2018, *arXiv:1807.03611*.

[97] K. Papadimitriou, A. Dollas, and S. Hauck, "Performance of partial reconfiguration in FPGA systems: A survey and a cost model," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 4, no. 4, pp. 1–36, 2011.

[98] S. Wang, Y. Liang, and W. Zhang, "FlexCL: An analytical performance model for OpenCL workloads on flexible FPGAs," in *Proc. 54th Annu. Design Automat. Conf.*, Jun. 2017, p. 27.

[99] E. Calore and S. F. Schifano, "Performance assessment of FPGAs as HPC accelerators using the FPGA empirical roofline," in *Proc. 31st Int. Conf. Field-Program. Log. Appl. (FPL)*, Aug. 2021, pp. 83–90.

[100] 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 *Proc. Perform. Modeling, Benchmarking Simul. High Perform. Comput. Syst. (PMBS)*, 2020, pp. 8–19.

[101] H. M. Makrani, F. Farahmand, H. Sayadi, S. Bondi, S. M. P. 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 *Proc. 29th Int. Conf. Field Program. Log. Appl. (FPL)*, Sep. 2019, pp. 397–403.

[102] F. Farahmand, A. Ferozpuri, W. Diehl, and K. Gaj, "Minerva: Automated hardware optimization tool," in *Proc. Int. Conf. ReConFigurable Comput. FPGAs (ReConFig)*, Dec. 2017, pp. 1–8.

[103] Z. Wang, B. He, W. Zhang, and S. Jiang, "A performance analysis framework for optimizing OpenCL applications on FPGAs," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Mar. 2016, pp. 114–125.

[104] C. Larman and V. R. Basili, "Iterative and incremental developments. A brief history," *Computer*, vol. 36, no. 6, pp. 47–56, 2003.

[105] J. Cong, W. Jiang, B. Liu, and Y. Zou, "Automatic memory partitioning and scheduling for throughput and power optimization," *ACM Trans. Des. Automat. Electron. Syst.*, vol. 16, no. 2, pp. 1–15, 2011.

[106] N. K. Pham, A. K. Singh, A. Kumar, and M. M. A. Khin, "Exploiting loop-array dependencies to accelerate the design space exploration with high level synthesis," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, 2015, pp. 157–162.

[107] B. Reagen, R. Adolf, Y. S. Shao, G.-Y. Wei, and D. Brooks, "MachSuite: Benchmarks for accelerator design and customized architectures," in *Proc. IEEE Int. Symp. Workload Characterization (IISWC)*, Oct. 2014, pp. 110–119.

[108] Y. Hara, H. Tomiyama, S. Honda, H. Takada, and K. Ishii, "CHStone: A benchmark program suite for practical C-based high-level synthesis," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2008, pp. 1192–1195.

[109] B. C. Schafer and A. Mahapatra, "S2CBench: Synthesizable SystemC benchmark suite for high-level synthesis," *IEEE Embedded Syst. Lett.*, vol. 6, no. 3, pp. 53–56, Sep. 2014.

[110] Y. Zhou, U. Gupta, S. Dai, R. Zhao, N. Srivastava, H. Jin, J. Featherston, Y.-H. Lai, G. Liu, G. A. Velasquez, W. Wang, and Z. Zhang, "Rosetta: A realistic high-level synthesis benchmark suite for software-programmable FPGAs," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, 2018, pp. 269–278.

[111] Q. Gautier, A. Althoff, P. Meng, and R. Kastner, "Spector: An OpenCL FPGA benchmark suite," in *Proc. Int. Conf. Field-Program. Technol. (FPT)*, Dec. 2016, pp. 141–148.

[112] B. Reagen, J. M. Hernández-Lobato, R. Adolf, M. Gelbart, P. Whatmough, G.-Y. Wei, and D. Brooks, "A case for efficient accelerator design space exploration via Bayesian optimization," in *Proc. Int. Symp. Low Power Electron. Design (ISLPED)*, 2017, pp. 1–6.

[113] C. Lo and P. Chow, "Model-based optimization of high level synthesis directives," in *Proc. 26th Int. Conf. Field Program. Log. Appl. (FPL)*, Aug. 2016, pp. 1–10.

[114] A. Mehrabi, A. Manocha, B. C. Lee, and D. J. Sorin, "Bayesian optimization for efficient accelerator synthesis," *ACM Trans. Archit. Code Optim.*, vol. 18, no. 1, pp. 1–25, Mar. 2021.

[115] N. Wu, Y. Xie, and C. Hao, "IronMan: GNN-assisted design space exploration in high-level synthesis via reinforcement learning," in *Proc. Great Lakes Symp. VLSI (GLSVLSI)*, Jun. 2021, pp. 39–44.

[116] M. Siracusa, E. Del Sozzo, M. Rabozzi, L. Di Tucci, S. Williams, D. Sciuto, and M. D. Santambrogio, "A comprehensive methodology to optimize FPGA designs via the roofline model," *IEEE Trans. Comput.*, vol. 71, no. 8, pp. 1903–1915, Aug. 2021.

[117] 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," *J. Parallel Distrib. Comput.*, vol. 133, pp. 407–419, Nov. 2019.

[118] 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 *Proc. 39th Int. Conf. Comput.-Aided Design*, Nov. 2020, pp. 1–9.

[119] R. Tessier and H. Giza, "Balancing logic utilization and area efficiency in FPGAs," in *Proc. 10th Int. Workshop Field Program. Logic Appl.*, vol. 1896, 2000, pp. 535–544.

[120] 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 Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 11, pp. 3736–3747, Nov. 2020.

[121] L. Piccolboni, P. Mantovani, G. D. Guglielmo, and L. P. Carloni, "COSMOS: Coordination of high-level synthesis and memory optimization for hardware accelerators," *ACM Trans. Embedded Comput. Syst.*, vol. 16, no. 5s, pp. 1–22, Oct. 2017.

[122] 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 *Proc. Design, Automat. Test Eur. Conf. Exhib. (DATE)*, 2016, pp. 918–923.

[123] S. Xu, S. Liu, Y. Liu, A. Mahapatra, M. Villaverde, F. Moreno, and B. Carrion Schafer, "Design space exploration of heterogeneous MPSoCs with variable number of hardware accelerators," *Microprocessors Microsyst.*, vol. 65, pp. 169–179, Mar. 2019.

[124] J. Kwon and L. P. Carloni, "Transfer learning for design-space exploration with high-level synthesis," in *Proc. Workshop Mach. Learn. (CAD)*, 2020, pp. 163–168.

[125] 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 *Proc. IEEE 26th Annu. Int. Symp. Field-Program. Custom Comput. Mach. (FCCM)*, Apr. 2018, pp. 129–132.

[126] S. Liu, F. C. Lau, and B. C. Schafer, "Accelerating FPGA prototyping through predictive model-based HLS design space exploration," in *Proc. 56th Annu. Design Autom. Conf.*, Jun. 2019, p. 97.

[127] A. S. B. Lopes and M. M. Pereira, "A machine learning approach to accelerating DSE of reconfigurable accelerator systems," in *Proc. 33rd Symp. Integr. Circuits Syst. Design (SBCCI)*, Aug. 2020, pp. 1–6.

[128] E. Ustun, C. Deng, D. Pal, Z. Li, and Z. Zhang, "Accurate operation delay prediction for FPGA HLS using graph neural networks," in *Proc. 39th Int. Conf. Comput.-Aided Design*, Nov. 2020, p. 87.

[129] M. Manuel, A. Kreddig, S. Conrady, N. A. Vu Doan, and W. Stechele, "Model-based design space exploration for approximate image processing on FPGA," in *Proc. IEEE Nordic Circuits Syst. Conf. (NorCAS)*, Oct. 2020, pp. 1–7.

[130] G. Zhong, A. Prakash, Y. Liang, T. Mitra, and S. Niar, "Lin-analyzer: A high-level performance analysis tool for FPGA-based accelerators," in *Proc. 53rd Annu. Design Automat. Conf.*, Austin, TX, USA, Jun. 2016, p. 136.

[131] A. B. Perina, J. Becker, and V. Bonato, "Lina: Timing-constrained high-level synthesis performance estimator for fast DSE," in *Proc. Int. Conf. Field-Program. Technol. (ICFPT)*, 2019, pp. 343–346.

[132] 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 *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2017, pp. 1141–1146.

[133] 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 Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 7, pp. 1428–1441, Jul. 2019.

[134] L. Ferretti, G. Ansaloni, and L. Pozzi, "Cluster-based heuristic for high level synthesis design space exploration," *IEEE Trans. Emerg. Topics Comput.*, vol. 9, no. 1, pp. 35–43, Jan. 2021.

[135] L. Ferretti, G. Ansaloni, and L. Pozzi, "Lattice-traversing design space exploration for high level synthesis," in *Proc. IEEE 36th Int. Conf. Comput. Design (ICCD)*, Oct. 2018, pp. 210–217.

[136] Q. Gautier, A. Althoff, C. L. Crutchfield, and R. Kastner, "Sherlock: A multi-objective design space exploration framework," *ACM Trans. Design Autom. Electron. Syst.*, vol. 27, no. 4, pp. 1–20, Jul. 2022.

[137] D. Koeplinger, R. Prabhakar, Y. Zhang, C. Delimitrou, C. Kozyrakis, and K. Olukotun, "Automatic generation of efficient accelerators for reconfigurable hardware," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2016, pp. 115–127.

[138] H. M. Makrani, H. Sayadi, T. Mohsenin, S. Rafatirad, A. Sasan, and H. Homayoun, "XPPE: Cross-platform performance estimation of hardware accelerators using machine learning," in *Proc. 24th Asia South Pacific Design Automat. Conf.*, Jan. 2019, pp. 727–732.

[139] *PowerTool MAXPOWERTOOL002 Quick Start Guide*, Maxim Integr., San Jose, CA, USA, 2014. Accessed: Feb. 19, 2022. [Online]. Available: https://pdfserv.maximintegrated.com/en/an/UG5981.pdf

[140] *USB Interface Adapter Evaluation Module. User's Guide*, Texas Instrum., Dallas, TX, USA, 2006. Accessed: Feb. 19, 2022.

[141] *Xilinx Power Estimator User Guide. UG-440 (v2021.2)*, Xilinx, San Jose, CA, USA, 2021. Accessed: Feb. 19, 2022. [Online]. Available: https://china.xilinx.com/content/dam/xilinx/support/documents/sw_manuals/xilinx2021_2/ug440-xilinx-power-estimator.pdf

[142] *Intel FPGA Power and Thermal Calculator User Guide*, Intel, San Jose, CA, USA, 2021. Accessed: Feb. 19, 2022. [Online]. Available: https://www.intel.com/content/www/us/en/docs/programmable/683445/21-4/overview-of-the.html

[143] J. J. Davis, E. Hung, J. M. Levine, E. A. Stott, P. Y. K. Cheung, and G. A. Constantinides, "KAPow: high-accuracy, low-overhead online per-module power estimation for FPGA designs," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 11, no. 1, pp. 1–22, Mar. 2018.

[144] S. Xu and B. C. Schafer, "Approximating behavioral HW accelerators through selective partial extractions onto synthesizable predictive models," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2019, pp. 1–8.

[145] J. Lorandel, J.-C. Prévotet, and M. Hélard, "Efficient modelling of FPGA-based IP blocks using neural networks," in *Proc. Int. Symp. Wireless Commun. Syst. (ISWCS)*, 2016, pp. 571–575.

[146] Y. Nasser, J. Prévotet, and M. Hélard, "Power modeling on FPGA: A neural model for RT-level power estimation," in *Proc. Int. Conf. Comput. Frontiers (CF)*, 2018, pp. 309–313.

[147] 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," *IEEE Embedded Syst. Lett.*, vol. 12, no. 2, pp. 58–61, Jun. 2020.

[148] G. Verma, V. Khare, and M. Kumar, "More precise FPGA power estimation and validation tool (FPEV_Tool) for low power applications," *Wireless Pers. Commun.*, vol. 106, no. 4, pp. 2237–2246, Jun. 2019.

[149] L. Deng, K. Sobti, and C. Chakrabarti, "Accurate models for estimating area and power of FPGA implementations," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, Mar. 2008, pp. 1417–1420.

[150] G. Verma, T. Singhal, R. Kumar, S. Chauhan, S. Shekhar, B. Pandey, and D. M. Akbar Hussain, "Heuristic and statistical power estimation model for FPGA based wireless systems," *Wireless Pers. Commun.*, vol. 106, no. 4, pp. 2087–2098, Jun. 2019.

[151] Y. Liang, S. Wang, and W. Zhang, "FlexCL: A model of performance and power for OpenCL workloads on FPGAs," *IEEE Trans. Comput.*, vol. 67, no. 12, pp. 1750–1764, Dec. 2018.

[152] 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 *Proc. Int. Conf. Comput.-Aided Design*, Nov. 2018, pp. 1–8.

[153] Z. Lin, J. Zhao, S. Sinha, and W. Zhang, "HL-Pow: A learning-based power modeling framework for high-level synthesis," in *Proc. Asia South Pacific Design Autom. Conf. (ASP-DAC)*, 2020, pp. 574–580.

[154] 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," in *Proc. Design, Automat. Test Eur. Conf. Exhib. (DATE)*, 2022, pp. 1341–1346.

[155] Y. S. Shao, B. Reagen, G.-Y. Wei, and D. Brooks, "Aladdin: A pre-RTL, power-performance accelerator simulator enabling large design space exploration of customized architectures," in *Proc. ACM/IEEE 41st Int. Symp. Comput. Archit. (ISCA)*, Jun. 2014, pp. 97–108.

[156] M. Makni, S. Niar, M. Baklouti, and M. Abid, "HAPE: A high-level area-power estimation framework for FPGA-based accelerators," *Microprocessors Microsyst.*, vol. 63, pp. 11–27, Nov. 2018.

[157] K. K. W. Poon, A. Yan, and S. J. E. Wilton, "A flexible power model for FPGAs," in *Proc. 12th Int. Conf. Field Program. Logic Appl. (FPL)*, vol. 2438, 2002, pp. 312–321.

[158] C. Du and Y. Yamaguchi, "High-level synthesis design for stencil computations on FPGA with high bandwidth memory," *Electronics*, vol. 9, no. 8, p. 1275, Aug. 2020. [Online]. Available: https://www.mdpi.com/2079-9292/9/8/1275

[159] 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 *Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, May 2021, pp. 1077–1086.

[160] K. Nagasu, K. Sano, F. Kono, and N. Nakasato, "FPGA-based tsunami simulation: Performance comparison with GPUs, and roofline model for scalability analysis," *J. Parallel Distrib. Comput.*, vol. 106, pp. 153–169, Aug. 2017.

[161] C. Du, I. Firmansyah, and Y. Yamaguchi, "FPGA-based computational fluid dynamics simulation architecture via high-level synthesis design method," in *Proc. Int. Symp. Appl. Reconfigurable Comput.*, vol. 12083, 2020, pp. 232–246.

[162] E. Reggiani, G. Natale, C. Moroni, and M. D. Santambrogio, "An FPGA-based acceleration methodology and performance model for iterative stencils," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops (IPDPSW)*, May 2018, pp. 115–122.

[163] M. Feickert and B. Nachman, "A living review of machine learning for particle physics," 2021, *arXiv:2102.02770*.

[164] A. M. C. Deiana, N. Tran, J. Agar, M. Blott, G. Di Guglielmo, J. Duarte, P. Harris, S. Hauck, M. Liu, M. S. Neubauer, and J. Ngadiuba, "Applications and techniques for fast machine learning in science," 2021, *arXiv:2110.13041*.

[165] S. L. Brunton, B. R. Noack, and P. Koumoutsakos, "Machine learning for fluid mechanics," *Annu. Rev. Fluid Mech.*, vol. 52, no. 1, pp. 477–508, Jan. 2020.

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

[167] A. Reuther, P. Michaleas, M. Jones, V. Gadepally, S. Samsi, and J. Kepner, "Survey and benchmarking of machine learning accelerators," in *Proc. IEEE High Perform. Extreme Comput. Conf. (HPEC)*, Sep. 2019, pp. 1–9.

[168] A. Reuther, P. Michaleas, M. Jones, V. Gadepally, S. Samsi, and J. Kepner, "Survey of machine learning accelerators," in *Proc. IEEE High Perform. Extreme Comput. Conf. (HPEC)*, Sep. 2020, pp. 1–12.

[169] 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 *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops (IPDPSW)*, May 2019, pp. 107–114.

[170] M. Motamedi, P. Gysel, V. Akella, and S. Ghiasi, "Design space exploration of FPGA-based deep convolutional neural networks," in *Proc. 21st Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2016, pp. 575–580.

[171] J. Xu, Z. Liu, J. Jiang, D. Yong, and S. Li, "CaFPGA: An automatic generation model for CNN accelerator," *Microprocess. Microsyst.*, vol. 60, pp. 196–206, Jul. 2018.

[172] 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," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 40, no. 2, pp. 301–314, Feb. 2021.

[173] S. O. Ayat, M. Khalil-Hani, and A. A.-H.-A. Rahman, "Optimizing FPGA-based CNN accelerator for energy efficiency with an extended roofline model," *Turkish J. Electr. Eng. Comput. Sci.*, vol. 26, no. 2, pp. 919–935, Mar. 2018.

[174] L. Xie, X. Fan, W. Cao, and L. Wang, "High throughput CNN accelerator design based on FPGA," in *Proc. Int. Conf. Field-Program. Technol. (FPT)*, Dec. 2018, pp. 274–277.

[175] 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. 172509–172523, 2020.

[176] Y. Ma, Y. Cao, S. Vrudhula, and J.-S. Seo, "Performance modeling for CNN inference accelerators on FPGA," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 4, pp. 843–856, Apr. 2020.

[177] T. Geng, T. Wang, A. Li, X. Jin, and M. Herbordt, "FPDeep: Scalable acceleration of CNN training on deeply-pipelined FPGA clusters," 2019, *arXiv:1901.01007*.

[178] 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 *Proc. IEEE 27th Int. Conf. Appl.-Specific Syst., Architectures Processors (ASAP)*, Jul. 2016, pp. 107–114.

[179] Y.-C. Lin, B. Zhang, and V. Prasanna, "HP-GNN: Generating high throughput GNN training implementation on CPU-FPGA heterogeneous platform," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, Feb. 2022, pp. 123–133.

[180] A. Ghaffari and Y. Savaria, "CNN2Gate: An implementation of convolutional neural networks inference on FPGAs with automated design space exploration," *Electronics*, vol. 9, no. 12, p. 2200, Dec. 2020.

[181] S. I. Venieris and C.-S. Bouganis, "FpgaConvNet: Automated mapping of convolutional neural networks on FPGAs," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, Feb. 2017, pp. 291–292.

[182] Y. Chen, J. He, X. Zhang, C. Hao, and D. Chen, "Cloud-DNN: An open framework for mapping DNN models to cloud FPGAs," in *Proc. Int. Symp. Field-Program. Gate Arrays*, 2019, pp. 73–82.

[183] 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 *Proc. IEEE Real-Time Syst. Symp. (RTSS)*, Nov. 2016, pp. 1–12.

[184] J. Mu, W. Zhang, H. Liang, and S. Sinha, "A collaborative framework for FPGA-based CNN design modeling and optimization," in *Proc. 28th Int. Conf. Field Program. Log. Appl. (FPL)*, 2018, pp. 139–1397.

[185] O. Reiche, M. A. Özkan, R. Membarth, J. Teich, and F. Hannig, "Generating FPGA-based image processing accelerators with Hipacc," in *Proc. Int. Conf. Computer-Aided Design (ICCAD)*, Nov. 2017, pp. 1026–1033.

[186] 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 *Proc. Design Autom. Conf. (DAC)*, 2018, p. 153.

[187] 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 *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, Feb. 2020, pp. 40–50.

[188] L. Ferretti, A. Cini, G. Zacharopoulos, C. Alippi, and L. Pozzi, "A graph deep learning framework for high-level synthesis design space exploration," 2021, *arXiv:2111.14767*.

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

[190] 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 *Proc. Annu. Int. Symp. Field-Program. Custom Comput. Mach.*, 2013, pp. 89–96.

ROMINA SOLEDAD MOLINA (Student Member, IEEE) received the master's (Master in Computer Science) degree from the Universidad Nacional de San Luis, Argentina. She is currently pursuing the Ph.D. degree in industrial and information engineering with the Università degli Studi di Trieste, under a Joint-Supervision Program with the Universidad Nacional de San Luis. Her main research interests include digital signal processing, digital control, image analysis, high-performance computing, machine learning, parallel computing models, FPGA, and SOC.

VERONICA GIL-COSTA is currently a Former Researcher at Yahoo! Labs Santiago hosted by the University of Chile. She is also an Associate Professor at the University of San Luis, a Researcher at the National Research Council (CONICET) of Argentina, and a Researcher at the CITIAPS, Chile. Her research work is on parallel computing and distributed systems, with applications in query processing and capacity planning for large scale systems.

MARÍA LIZ CRESPO is currently a Research Officer at The Abdus Salam International Centre for Theoretical Physics (ICTP) and an Associate Researcher of the Italian National Institute of Nuclear Physics (INFN), Trieste, Italy. She is also coordinating the research and training program of the Multidisciplinary Laboratory (MLab), ICTP. She has organized several international schools and workshops on fully programmable systems on chip for nuclear and scientific instrumentation. She is the coauthor of more than 100 scientific publications in prestigious peer-reviewed journals. Her main research interests include advanced scientific instrumentation for particle physics experiments and experimental multidisciplinary research.

GIOVANNI RAMPONI (Life Senior Member, IEEE) was born in 1956. Since 2000, he has been a Full Professor of electronics at the Department of Engineering and Architecture, University of Trieste, Italy. He is the co-inventor of international patents, and has published more than 200 papers in international journals, conference proceedings, and book chapters. His research interests include nonlinear digital signal processing, enhancement and feature extraction in images and image sequences, image visualization, image quality evaluation, and deep learning techniques for image processing. More information can be found at: www.units.it/ramponi.