Accepted for IEEE EDGE 2023

DOSA: Organic Compilation for Neural Network Inference on Distributed FPGAs

Burkhard Ringlein[*][†], François Abel[*], Dionysios Diamantopoulos[*], Beat Weiss[*], Christoph Hagleitner[*], and Dietmar Fey[†]
[*]IBM Research Europe, [†]Friedrich-Alexander University Erlangen-Nürnberg
{ngl, fab, did, wei, hle}@zurich.ibm.com, {burkhard.ringlein, dietmar.fey}@fau.de

***Abstract*—The computational requirements of artificial intelligence workloads are growing exponentially. In addition, more and more compute is moved towards the edge due to latency or localization constraints. At the same time, Dennard scaling has ended and Moore's law is winding down. These trends created an opportunity for specialized accelerators including field-programmable gate arrays (FPGAs), but the poor support and usability of today's tools prevents FPGAs from being deployed at scale for deep neural network (DNN) inference applications. In this work, we propose an organic compiler — DOSA — that drastically lowers the barrier for deploying FPGAs. DOSA builds on the operation set architecture concept and integrates the DNN accelerator components generated by existing DNN-to-FPGA frameworks to produce an overall efficient solution. DOSA starts from DNNs represented in the community standard ONNX and automatically implements model- and data-parallelism, based on the performance targets and resource footprints provided by the user. Deploying a DNN using DOSA on 9 FPGAs exhibits a speedup of up to 52 times compared to a CPU and 18 times compared to a GPU.**

***Index Terms*—MLSys, Reconfigurable hardware, Domain-specific architectures, Compilers, Distributed Artificial Intelligence, Design Tools and Techniques**

I. Introduction

Today, domain-specific architectures are developed and deployed to address the exponential increase of the computational demands of machine learning (ML) and artificial intelligence (AI) applications. This more hardware-centric approach was triggered by the slow down of technology scaling [1] and the drastic increase in complexity of AI models and their use cases [2]. Likewise, AI and ML applications are now ubiquitous and push data processing to the edge, due to latency or power constraints [3]–[5]. Furthermore, regulation and privacy considerations impose additional design restrictions by limiting the localization and movement of data. The combination of these three trends led to a "Cambrian explosion" [6] of new AI accelerators[1]. While many of these accelerators are GPUs or application-specific integrated circuits (ASICs), field-programmable gate arrays (FPGAs) became also increasingly popular. FPGAs offer an attractive "middle ground" between the lack of energy-efficiency of general-purpose GPUs and CPUs and the lack of adaptability of full-custom ASICs. They are energy efficient, can provide superior performance [6]–[9] and adapt to frequent ML model changes [2], [10]. Therefore, FPGAs are widely used for energy- and latency-constrained ML applications at the edge [4], [5], [11]–[17].

[1] **Terminology:** In this work, the term **topology** refers to the structure of a neural network, i.e. its internal structure of operations or layers. **Architecture** refers to the hardware implementation, and its properties, of a neural network topology. **Accelerator** refers to the subset of an architecture (or to the complete architecture). **Framework** refers to a set of scripts used to turn a topology into an architecture (e.g. hls4ml [18]). **Tool flow** (or tool chain) refers to a set of tools used to map (i.e. compile, synthesize, place and route) an accelerator to an FPGA device.

[2] https://github.com/cloudFPGA/DOSA

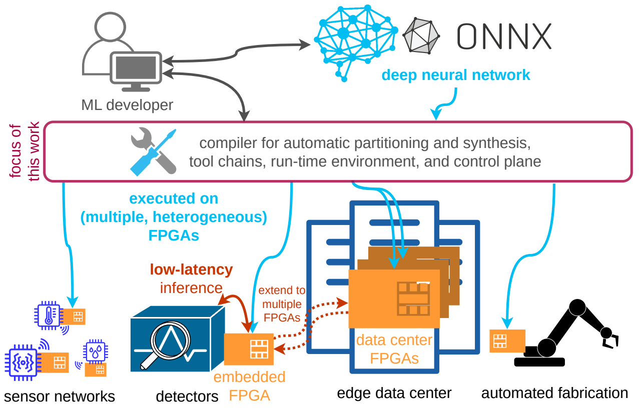


Fig. 1. Deploying low-latency DNN inference at the edge using FPGAs.

A. Motivational Example

Figure 1 shows an illustrative example of a depp neural networks (DNNs) application deployed for a low latency inference — 1 ms or below — classification at the edge. The model for the inference service is developed by an ML expert, who has little or no FPGA expertise, and is specified in a widely supported community standard, such as the Open Neural Network eXchange (ONNX) [19]. If the DNN model is too large to fit in the FPGA at the data source, the (remaining part of the) DNN can be executed in a nearby edge data center. Hence, larger DNN models may expand to multiple devices. Despite their favorable performance and energy efficiency, such an FPGA implementation solution must also offer competitive development and deployment complexity compared to GPU or CPU based solutions. Hence, the goal of our research is to develop a tool flow capable of turning an ONNX model into a distributed FPGA design, in just "one-click".

B. Problem Statement and Contributions

Today's DNN-to-FPGA frameworks support only a very narrow set of DNN topologies and target devices and "coding even simple algorithms on FPGAs remains very painful and time-consuming" [10]. In addition, deploying large AI models on multiple FPGAs is difficult, because most tool-flows do not support partitioning.

In this work, we first analyze the reasons behind the limited adoption of FPGAs for AI inference at the edge. We use our recently proposed organic compilation concept, which leverages the large body of FPGA-optimized building blocks generated by exiting tools [20]. We present DOSA[2], an open source implementation of the proposed organic compiler, which supports deployment on distributed FPGAs. In particular, the main contributions of this research are:

1) A tool to automatically distribute a DNN specified in ONNX to multiple heterogeneous nodes within seconds, supporting model and data parallelism leveraging organic compilation.

This is the accepted version of the article published by IEEE: 10.1109/EDGE60047.2023.00019

2) A unified evaluation criterion for DNN operations to find the optimum in a heterogeneous design space.
3) The automated generation of host-to-FPGA, inter-FPGA and intra-FPGA communication and the corresponding software run-time environment.
4) The demonstration of an end-to-end example, running on multiple FPGAs and outperforming CPUs and GPUs.

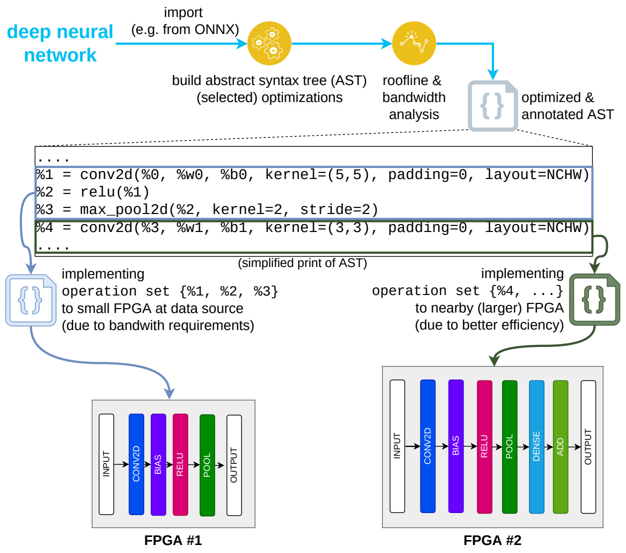
DOSA enables the compilation, deployment, and execution of a DNN across distributed FPGAs with speedup gains of up to 50 times compared to a CPU and 18 times compared to a GPU. The paper is structured as follows: Section II introduces the motivation behind organic compilation and discusses prior art. This is followed by the presentation of our organic compiler DOSA in Section III. Finally, we evaluate our framework in depth and demonstrate an multi-FPGA end-to-end example before we provide a conclusion.

II. Motivation and Related Work

We reviewed the landscape of frameworks that can help a user to deploy a DNN to an FPGA, and we found a significant amount of research tools [7]–[9], [13], [18], [21]–[27]. However, despite the availability of such frameworks, FPGAs are seldom used outside of their community because they are considered difficult to program [10], [20]. We attribute this lack of adoption to four problems with the current state of the art tools: First, despite the vast range of options for DNN-to-FPGA flows, all frameworks support only a limited range of DNN operations, or a very narrow set of target devices, or both limitations are given [28], [29]. Second, there exists no guide to help the user select the right framework for a given DNN and application scenario, let alone an automated tool to do so [20]. For example, the tool *SAMO* [25] improves the efficiency of the considered specialist-frameworks by adding common optimization passes to all of them, but does not combine different accelerators or help the user in selecting the right framework for a given problem. Likewise, existing benchmarks like *ECBA-MLI* [30], which evaluates latency and power consumption of different DNN deployments at the edge, do not support FPGAs. Third, compatibility between tool flows, frameworks and vendor tools is very limited. In addition, a design flow breaks frequently, if just one of the involved tools is upgraded to a newer (minor) version. Lastly, most of today's DNN-to-FPGA frameworks are limited to single device use cases and can not distribute a given DNN across multiple FPGAs, which limits the size and throughput of the compiled DNN models.

Given the richness of existing optimized but narrow frameworks, we envision a DNN compiler that can reuse and combine these existing frameworks as one tool. If a certain framework can already implement a specific part of a DNN in an optimized way, e.g. the haddoc2 framework [22] can synthesize 2D convolutions with certain fixed-point data types to a high-throughput architecture, why not reuse this work and avoid to "re-invent the wheel" again?

The foundation of organic compilation is to combine multiple specialized-but-narrow frameworks into a single holistic compiler. This requires a compiler that can reason about different parts of the DNN, different devices, and different frameworks in an unified way. To achieve this goal, we build on our previously proposed *Operation Set Architecture* (OSA) [20], a concept that enables the architecture-agnostic optimization of a DNN and subsequently the selection of the best implementation among multiple possibilities. This basic principle is shown in Figure 2. Using the OSA, an abstract syntax tree (AST) is built from the imported DNN and simple


deep neural network
import (e.g. from ONNX)
build abstract syntax tree (AST) (selected) optimizations
roofline & bandwidth analysis
optimized & annotated AST
(simplified print of AST)
implementing
to small FPGA at data source
(due to bandwith requirements)
implementing
to nearby (larger) FPGA
(due to better efficiency)
INPUT
CONV2D
BIAS
RELU
POOL
OUTPUT
FPGA #1
INPUT
CONV2D
BIAS
RELU
POOL
DENSE
ADD
OUTPUT
FPGA #2

```
....
%1 = conv2d(%0, %w0, %b0, kernel=(5,5), padding=0, layout=NCHW)
%2 = relu(%1)
%3 = max_pool2d(%2, kernel=2, stride=2)
%4 = conv2d(%3, %w1, %b1, kernel=(3,3), padding=0, layout=NCHW)
....
```

```
operation set {%1, %2, %3}
```

```
operation set {%4, ...}
```

Fig. 2. Basic principle of the distributed Operation Set Architecture (OSA).

optimizations are executed. Additionally, the AST is lowered to an intermediate representation (IR) that allows to handle small-enough but still "meaningful" operations. In Figure 2, these are the operations `conv2d`, `relu`, `max_pool2d`, which abstracts certain parts of the topology of the imported DNN. The level of abstraction used by OSA is comparable to the abstraction levels used by popular domain-specific languages (DSLs) like RelayIR [31] or ONNX [32] or some MLIR dialects [33]–[36] (cf. also [37]). In fact, also many of the already existing DNN-to-FPGA frameworks provide IP cores on this level of abstraction. For example, the academic frameworks like *hls4ml* [13], [18], [21], *haddoc2* [22], *FINN* [7]–[9], or *AIgean* [23] provide interfaces to implement (some kind of) a `conv2d`. Hence, taking architectural decisions on this proper level of abstraction avoids a complex and error-prone mapping of low-level instructions to an FPGA IP core (cf. polyhedral-based compilation techniques [38], [39]). Additionally, the abstraction level provided by OSAs enable partitioning decisions at likewise "meaningful" steps of the application. This allows the implementation of a fast DSE. In the example of Figure 2, the roofline and bandwidth and analyses (cf. Figures 3 and 4) indicates a partitioning after the `max_pool2d` operation, as depicted at the lower half of the Figure 2.

To not exceed the scope of this section, we refer the reader to [20] for more background on OSAs and to [28], [29], or [40] for a detailed review of existing DNN-to-FPGA frameworks.

III. DOSA: Distributed Operation Set Architecture — Or how to Automate Reuse for DNNs on FPGAs

Our goal is to develop an "organic compiler" that analyzes a given DNN and selects the best-possible template-type and implementation offered by a number of frameworks for each arithmetic operation of this DNN. This compiler should understand a conventional DNN exchange standard and provide the user with insights on achievable performance, possible bottlenecks, and how the user's constraints influence the architectural decision. The design-space exploration (DSE) by this compiler should also consider partitioning, with model- and device-parallelism as options, and should ideally take only a few seconds, to allow frequent iterations and optimizations with a user in the loop. Therefore, this compiler must be able to predict performance and resource consumption for each possible implementation quickly and reliably and must use meaningful criteria to compare these estimates. In this section,

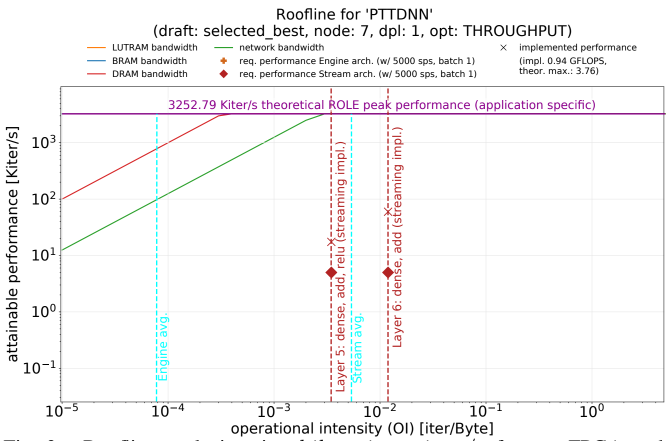


Fig. 3. Roofline analysis using $kilo - iterations/s$ for one FPGA node (pipeline stage 5 of Figure 6).

we present the important ingredient of a unified evaluation criterion, before we discuss every step of the flow of an organic compiler, as shown in Figure 5.

A. A Unified Evaluation Criterion: Replacing FLOP/s on FPGAs

The core feature of our organic compiler for DNNs is its ability to automatically select from different specialist-frameworks the best mix of implementations on a mix of devices for a given application. This ability requires not only the "packing" of different implementations into the same categories, hence the selection of a DSL IR as done by the OSA, but also the possibility to evaluate two semantically identical, but differently implemented operations, based on predicted resource consumption or performance characteristics. Therefore, unified measurements to compare these numbers are needed, to be able to select the most efficient implementation available.

One natural criterion to compare the performance of different hardware implementations and to judge their effectiveness is to calculate their achieved $\frac{FLOPs}{s}$ (or just FLOPS), i.e. a measurement of how many arithmetic (floating point) operations can be done in one second. The notion of FLOPS historically originates from CPUs and their floating point units or central arithmetic logic units, but it is also used for FPGAs to compare their performance in marketing [41] and research [42]. It also appears to be a useful measurement for the purpose of our compiler, since we know what the necessary arithmetic steps for e.g. a 2D convolutions are, and we know the size and used data types of all inputs and parameters. Hence, we can calculate the necessary FLOPS of each operation and then, based on the architecture type, infer how long each available implementation would take as well as how many FPGA resources would be consumed. This would fit well with our roofline analysis, typically based on FLOPS, since it is the established way to determine the potential performance of a given algorithm on a given hardware and to identify potential performance bottlenecks (cf. [20]).

However, after building our compiler with this approach, we noticed that the training of a prediction function like

$$f(\text{FLOPS},\ \text{operation},\ \text{framework}) \rightarrow \text{resource usage},\ \text{latency}$$

is very difficult and in most cases quite inaccurate, even if we consider the detailed operation, data type, and sparsity of the weights, as well as using conservative approaches and not (marketing) data sheets (cf. [43]). As a short demonstration, we show in Table I selected examples of addition, subtraction, multiplication, division, or comparison, which could all be seen as one FLOP or integer operation (IOP). Even for this

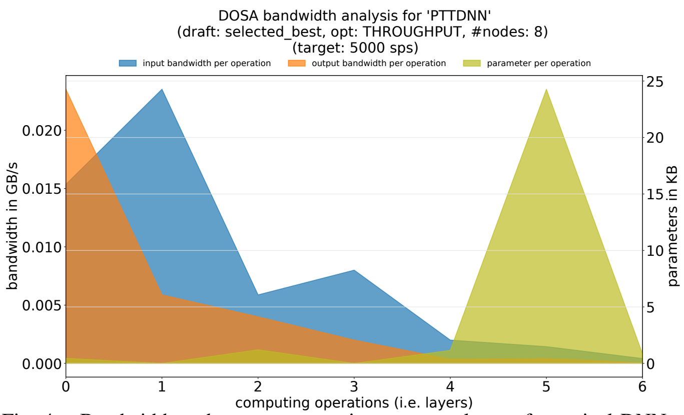


Fig. 4. Bandwidth and parameter requirements per layer of a typical DNN. The used topology is equivalent to the one in Figure 6.

very small sample, the deviation between the implementations is significant. Also, most of these operations do not use digital signal processors (DSPs), as would be expected or is indicated by some data sheets. After some investigation of this behavior, we concluded that measuring the performance and efficiency of reconfigurable hardware, with its core features of customization of the data path and control flow for each algorithm, using a metric, which is supposed to express the performance of a fixed hardware on all kind of computations, is a conceptually flawed approach. Due to the wide design space of a single arithmetic operation, from one LUT for 1-bit multiplication up to multiple DSPs and hundreds of LUTs for larger floating point multiplications, on one side and on the other side the "fuzziness" of a "FLOP/IOP", the prediction of resource usage and performance using FLOPS is not suitable for FPGAs. For example, how many FLOPS does a max pooling operation, which performs only comparisons, use? Or how to account for the data paths that are necessary to bring the data to these comparators (cf. [44])? Such questions must be answered by a compiler during a DSE phase. This misfit of concepts is also not solved by measuring a few representative example implementations on the targeted FPGA, calculating their FLOPS and taking these measurements as a baseline (cf. [42]), since all the different possible optimizations for a slightly different algorithm or even just a different parameter lead to high uncertainty again.

Consequently, we needed to find a different measure that may not be as general as FLOPS but more accurate and more helpful. Hence, we decided to use $\frac{iterations}{s}$, *for one specific unit under consideration*. For example, when comparing two max pooling implementations for the same operation the compiler can compare both implementations in terms of possible iterations per second and resource consumption. Based on this throughput calculation, the latency can be derived directly, if this is the optimization goal of interest. Similarly, when comparing multiple operations on different FPGAs, the compiler can easily determine what the total $iterations/s$ for each node is, since the total performance depends on the slowest operation and possible parallel executions. Finally, implementations of complete DNNs can also be compared, since the total iterations per second can be calculated likewise by selecting the iterations of the bottlenecks.

While this may sound like a minor difference to using FLOPS, the implications are substantial. First, the used $iterations/s$ numbers are always bound to specific domain-specific instructions and therefore more precise or "meaningful". For example, the performance roof of the roofline in Figure 3 applies to the set of operations scheduled on this particular FPGA. Second, how long one iteration takes can

TABLE I
EXCERPTS FROM THE RESOURCE USAGE OF ONE-FLOP/IOP-LIKE OPERATIONS ON KINTEX ULTRASCALE FPGAS [45]–[47].

Operation	Resource usage					
	DSPs	FFs	LUTs	BRAMs (36k)	Fmax (MHz)	latency (cycles)
32 bit int add/sub	0	104	36	0	604	3
	1	0	1	0	631	2
32 bit uint add/sub	0	106	59	0	571	3
28 bit const. coeff. int mult.	0	68	121	3.5	286	3
35 bit int mult.	4	69	18	0	631	6
12 bit uint mult.	0	173	125	0	467	3
32 bit float add/sub	2	311	189	0	539	N/A
	0	578	349	0	643	
32 bit float comparison	0	12	48	0	1,155	
32 bit int to float conversion	0	228	157	0	635	
32 bit float mult.	2	166	90	0	568	
	3	123	73	0	594	
	0	695	570	0	590	
32 bit float division	0	1,383	763	0	598	
1 bit mult. on Zynq [9]	0	1	1	0	>200	N/A

be easily measured in hardware or by analyzing simulations. Hence, the measure is not as fuzzy as FLOPS. Third, the measured iteration accounts for all hardware that is necessary to perform the specified operation on the specified inputs, e.g. it counts the necessary FIFOs for buffering, logic for the control-flow, or all needed comparators. Finally, we noticed after some experiments that performance and resource usage predictions based on existing measurements of iterations (i.e. $\frac{frequency}{latency}$) for a specific size of inputs and parameters are more accurate than using FLOPS. This unit of measurement is also compatible with the our idea to use a roofline analysis as basis for the DSE. The meaning of the OI in this type of roofline analysis is the same: The higher the intensity, the fewer bytes are used for one iteration. Hence, the "bandwidth-barriers" are still valid (cf. Figure 3). Subsequently, we decided to use $\frac{iterations}{s}$ as criterion when comparing the performance of different available implementations of one specific operation in the AST.

B. Flow of an Organic Compiler for Distributed FPGAs

Figure 5 shows the flow diagram of our organic compiler. Such an organic compiler requires a DSE phase that can analyze the DNN and which knows about the characteristics of the available frameworks. Consequently, an organic compiler has four inputs: The DNN Ⓐ, specified in a community standard as e.g. ONNX [19], the targeted performance and resource constraints Ⓑ, the description of the targeted devices Ⓔ, and the available specialist frameworks as library Ⓓ.

The flow starts ① with the import of the DNN and the execution of straightforward optimizations, such as constant folding, dead code elimination or operator fusion. Also, an AST of the DNN is built. In parallel, the library of specialist DNN-to-FPGA frameworks Ⓓ and the library of available target platforms Ⓔ are imported ② and prepared for the DSE using the evaluation criterion developed in the previous section $\frac{iterations}{s}$ Ⓒ. In the next step ③, the characterizations of step ② are then used to annotate the AST of the DNN operation-wise using a roofline-like analysis (cf. Figure 3), together with the library of available platform characterizations.

Afterwards, having a detailed AST annotation, the DSE phase starts with partitioning the DNN ④, if required by the size or throughput requirements of the DNN. The partitioning is based on the roofline-analysis (see Figure 3) and bandwidth-analysis (see Figure 4). Next, based on an updated roofline analysis and the estimated latencies between nodes, high-level architectural decisions are made ⑤. Foremost, this decision involves to decide if weights of the operations of the DNN can be stored in an off-chip memory or if it has to stay on-chip, because the available bandwidth would not allow

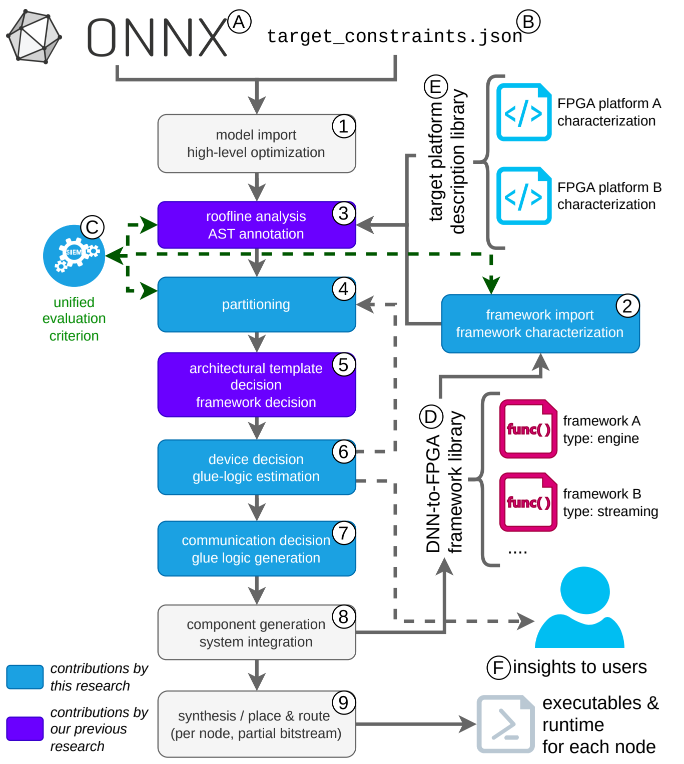

ONNX Ⓐ
target_constraints.json Ⓑ
model import high-level optimization ①
roofline analysis AST annotation ③
Ⓒ unified evaluation criterion
partitioning ④
architectural template decision framework decision ⑤
device decision glue-logic estimation ⑥
communication decision glue logic generation ⑦
component generation system integration ⑧
synthesis / place & route (per node, partial bitstream) ⑨
Ⓔ target platform description library
FPGA platform A characterization
FPGA platform B characterization
framework import framework characterization ②
Ⓓ DNN-to-FPGA framework library
framework A type: engine
framework B type: streaming
....
Ⓕ insights to users
executables & runtime for each node
contributions by this research
contributions by our previous research

Fig. 5. Flow of an organic compilation to (distributed) FPGAs.

to load them fast enough (cf. [20]). Also, the best available specialist framework that can implement the decided micro-architecture with the derived performance requirements is selected. Next ⑥, if multiple target devices are available, the best candidates are selected in this step. This step also includes to calculate the resources necessary for the "glue logic" between the selected accelerator blocks. Here, it could be that this glue logic consumes more resources that are left on some devices. In that case, this result is annotated and the compiler continues with another partition step. After a valid solution is found, or in case the compiler fails to find one, the user is informed about the resulting performance, resource footprints, and potential bottlenecks Ⓕ. As seventh step ⑦, the details of the communication between FPGA nodes are decided, if the solution consists of multiple nodes. This involves finding the best synchronization pattern and deciding the type of serialization of multi-dimensional tensors. These decisions influence the latency between FPGA nodes only minimally, since they are all implemented in a data-flow architecture as part of the network stack of the FPGA logic [48], [49]. For the inter-node communication, DOSA builds on the ZRLMPI framework [50], [51]. This framework provides FPGA cores and CPU software to synchronize FPGA and CPU nodes at run-time by implementing a subset of the MPI standard. Therefore, DOSA instantiates the necessary Message Passing Engines from the ZLRMPI framework and connects them to the accelerator cores within the FPGA node. Additionally, this network adapter core has the communication plan of the node as table. This communication plan is generated by DOSA and applies all parallelizations that were inserted by the DSE, i.e. horizontal and vertical model parallelism and data parallelism. Here, also the abstraction level of the OSA helps, since the communication pattern of the operations can be derived in a straightforward way at this IR level (cf. Figure 2). For non-byte aligned data-types, on top of ZRLMPI, DOSA can leverage the PHRYCTORIA framework as a messaging system, which supports dynamically-adapted mixed-precision workloads for transprecision [52]. Lastly, the network streams and memory

TABLE II
DNN-TO-FPGA FRAMEWORKS: PRODUCTIVITY ANALYSIS.

Framework	supports ONNX import	supports distrib. FPGAs	manual scheduling or partitioning required	automated deployment
DOSA *(this research)*	yes	yes	no	yes
AIgean [23]	no	yes	no	no
hls4ml [13], [21]	yes	no	no	no
haddoc2 [22] req. legacy BVLC-Caffe	no	no	no	no
Brevitas + FINN [8], [9], [53]	no	(up to 2) [54]	partly	partly
VitisAI [55]	no	no	depends on the model	partly

TABLE III
EXAMPLE TOPOLOGIES FOR EVALUATION (ALL USING 8BIT WEIGHTS).

Task	Network topology	# Conv.	# Dense	Parameter (KB)
Jet Tagging	CERN 3 layer	0	4	4.4
Hand Gestures	MPCNN	3	2	70.3
MNIST	TFC	0	4	59.2
	LeNet-5	3	2	32.6
CIFAR-10	PTTDNN	3	2	32.6

buses are connected to the corresponding interfaces offered by the Shell of the selected target platforms (cf. [49], [51]).

Finally, if all decisions are made, the specialist frameworks are called to generate the desired building blocks ⑧. Additionally, the HDL of the glue logic is emitted, and all components are integrated into top-level HDL design files. This step includes the generation of the corresponding software run-time environment to manage the resulting cluster of FPGAs. As last step ⑨, synthesis, place and route is executed per FPGA node. In this work, the presented DSE phase assumes that the weights of the input topology are already quantized. While an automated optimization of different data-types throughout the DNN topology would be feasible and sometimes necessary [7], [24], [26], [56], [57], it is out of scope for this research.

IV. EVALUATION

In this Section we evaluate and quantify the benefits of DOSA. We start by comparing the productivity-gains of DOSA in contrast to existing frameworks. Second, we evaluate the coverage of a model zoo by different OSGs and their combination. Third, we analyze the DSE on the basis of multiple DNNs. Lastly, we demonstrate the use of DOSA to deploy an AI inference application across multiple FPGAs.

A. Productivity

One of our main goals is lowering the barrier for the deployment of FPGAs by non-FPGA experts. We measure our progress by comparing our developed tool with other DNN-to-FPGA tools offered by academia and industry in Table II. DOSA has a simple command line interface to automatically compile, build and deploy a DNN on a heterogeneous, distributed cluster. Due to the roofline, bandwidth, parameter, and device-compatibility analysis of DOSA, the scheduling and partitioning is possible completely without any involvement of the user. Alternative frameworks like FINN or VitisAI require the user to program in python, C++, and HLS to adapt the accelerator to the user's needs. Hence, without DOSA, a user who wants to deploy such a DNN is required to manually identify which part is supported by which framework, partition the DNN, generate the partial designs, and — depending on the framework — write the necessary glue logic manually. DOSA automates this completely and creates required build and deploy scripts. Furthermore, DOSA supports the very popular community standard ONNX. Table II summarizes some important features that aid user's productivity. In summary, DOSA improves the productivity of the user significantly and offers better coverage of ONNX than comparable frameworks.

TABLE IV
OPERATION COVERAGE OF DIFFERENT DNN-TO-FPGA FRAMEWORKS.

Example	Operation coverage of individual frameworks			
	haddoc2	hls4ml	TIPS	DOSA *(the combination, this research)*
CERN 3 layer	0.27	1.0	1.0	1.0
MPCNN	0.76	0.82	0.52	1.0
TFC	0.27	1.0	0.90	1.0
LeNet-5	0.76	0.88	0.58	1.0
PTTDNN	0.81	0.81	0.19	1.0

TABLE V
RESULTS OF DOSA DSE

Example	Number of FPGA nodes	Predicted throughput (iops)	DSE time (s)
CERN 3 layer	1	78,397.17	0.89
MPCNN	15	9,945.26	0.11
TFC	1	6,399.77	0.78
LeNet-5	18	5,066.91	0.20
PTTDNN	9	17,421.59	0.08

B. Coverage of DNN Operations by Combining Frameworks

The key motivation behind developing DOSA was to be able to combine different specialist DNN-to-FPGA frameworks and leverage this existing expertise. We observed that most of the frameworks support only a very narrow set of DNNs, especially academic tools like haddoc2, hls4ml or TIPS [20]. For example, the PTTDNN, which is discussed in Section IV-D, can't be implemented by either haddoc2 nor hls4ml alone, since both do not support all necessary operations. To quantify this observation, Table IV shows the share of operations that can be implemented by each framework for a number of DNNs. The DNN topologies and their application areas are specified in Table III. The combination of these specialist frameworks by DOSA can cover all necessary operations, as is shown in the last column of Table IV. Hence, DOSA does not only improve the productivity, but significantly increase the topologies that can be implemented on FPGAs.

C. DOSA Design-Space Exploration

The primary use-case for DOSA are high-throughput or low-latency inference applications, potentially in a heterogeneous edge environment (cf. Figure 1). Therefore, we evaluated the generated architectures for a number of topologies for this application domain (cf. examples in Section I-A). For each of these DNNs, DOSA was invoked to generate a distributed FPGA design with the goal to achieve at least 5000 iops (inference-operations per second). After determining the minimum number of FPGAs required to achieve the performance goal, DOSA maximizes the throughput by fully utilizing the FPGA nodes, even if this exceeds the performance goal. The results of this analysis are printed in Table V. In all cases, the performance goal is achieved. Additionally, the DSE including the generation of all FPGA designs, build scripts, and software run-time environment, takes less than 1 s, which is significantly faster then related state of the art (cf. [54]).

D. Using DOSA for High-throughput Low-latency Inference

Finally, we demonstrate DOSA end-to-end with one DNN for classification of the `CIFAR-10` data set using the `PTTDNN` topology (PyTorch Tutorial DNN, based on [58], see Table III) and use this application to discuss all aspects of the DSE for partitioned DNNs in depth. Figure 6 shows how the CIFAR-10 DNN is partitioned across nine FPGAs. We use the IBM** cloudFPGA platform [48], [49], [59], [60] as testbed for distributed edge FPGA environments. This platform consists of disaggregated Kintex KU060 FPGAs (`xcku060-ffva1156-2-i`), each with a direct 10 GbE network attachment. The Shell of the FPGAs consumes approximately 50 % of the total FPGA. Hence the part that is available for use by DOSA are up to 40 % of the resources

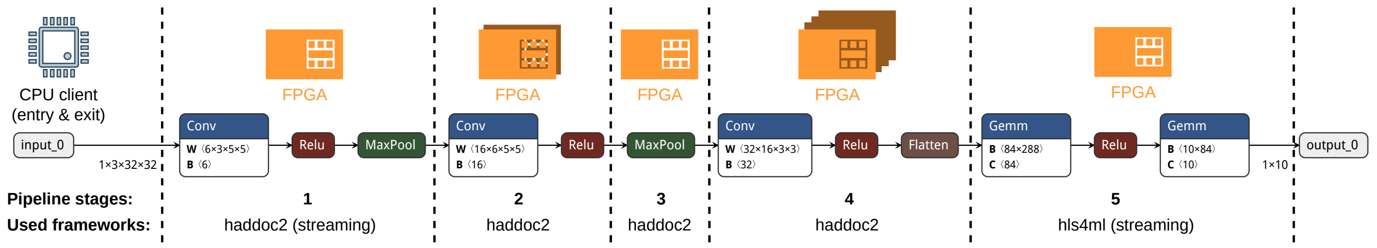


Fig. 6. One example of a DNN spread (PTTDNN) across 9 FPGAs. The CPU client just performs send and receive calls.

TABLE VI
RESULTS OF THE PTTDNN INFERENCE SERVICE ON THREE PLATFORMS.

Devices	Measured throughput (iops)	End-to-end latency (ms)	Average power (W)	Total energy per inf. (J)
9x KU060 FPGAs (156 MHz)	3,853.25*	0.259	77.31	0.020
1x Xeon E5-2630 (2.4 GHz)	73.38	13.627	123.69	1.686
1x Tesla K40c (745 MHz)	211.81	4.721	129.51	0.676

*: Limited by the single-socket CPU client.

of a Kintex KU060, since a target resource utilization >90 % lead to routing or timing failures. To compare the inference service offered by standalone FPGAs with CPUs, we deployed PTTDNN using pytorch on an Intel‡ Xeon E5-2630 v3 CPU with 8 cores and 126 GB memory running Ubuntu 20.04. We also measured the same service using a Nvidia Tesla K40c (`GK180GL`). We have selected devices of comparable technology nodes (CPU 22 nm, FPGA 20 nm, GPU 28 nm). Since more and more embedded and edge components are connected via the standard "internet stack" TCP/UDP/IP [11], [12], [18], we use integrated network interfaces for inter-FPGA communication and the connection to the services, which send the source data and collect the results. The batch size for all experiments was set to one as demanded by the low-latency inference applications described in Section I-A, e.g. for an industrial vision classification. The CPU client is not counted towards the power consumption in all cases. The results of our experiments are shown in Table VI.

First, this example shows the value of DOSA, because this frameworks enables multi-FPGA inference with "one click". The partitioning across FPGAs, glue logic writing, communication implementation, and software run-time generation is completely automated. We estimate that an experienced FPGA developer would require roughly one or two months to perform the same tasks manually with the current flows available, as shown in Section II. Next, the results in Table VI exhibit the potential of disaggregated network-attached FPGAs for low latency classification applications. The inference requests to the DNN distributed across nine FPGAs are served within 0.3 ms, 52 times faster compared with a CPU-based service, and 18 times faster compared with the GPU. Likewise, the FPGAs are at least 84 times more energy efficient compared to the CPU and at least 30 times more energy efficient than a GPU. This highlights also the advantage of disaggregation, since the GPU alone would consume roughly 19 W but needs a CPU attached to respond to network requests or to connect with other data sources. Furthermore, the throughput of the nine FPGA example in Figure 6 is close to 4000 iops, with the measured peak throughput of the slowest pipeline stage (stage 5 in the figure) at 17,970 iops. This exceeds the required 5000 iops by far and highlights the fact that performance and resource consumption are not correlated linearly. For example, a design for PTTDNN with a performance goal less than 1000 iops would require only one FPGA. To achieve such a high throughput, the DNN is completely implemented with streaming architectures, i.e. haddoc2 and hls4ml. Despite the all-streaming architecture, the advantage of combining different operators can be seen here as well: The first layers, i.e. pipeline stages 1 – 4 in Figure 6, are implemented without any reuse using the haddoc2 framework, i.e. every weight has a dedicated multiplication or addition circuit. Consequently, the network must be partitioned across nine FPGAs, despite the relatively small number of parameters. The scheduling of only one max pooling operation on FPGA of pipeline stage 3 in Figure 6 appears to be sub-optimal, but this operation can't be added to the FPGAs of stage 2 or 4, since they are fully consumed by the convolutional operations. Hence, DOSA would select larger FPGAs for this stages, but those aren't available in our current testbed. The last layer is implemented using hls4ml with a reuse factor of 32. This results in lower throughput, which is still sufficient due to the lower required bandwidth at the end of the DNN (cf. Figure 4). Additionally, haddoc2 can't implement dense operations while hls4ml would consume a lot more resources for the same convolutions. As seen, combining two different OSGs results is a more efficient solution.

V. CONCLUSION

Boosting the adoption of FPGAs for AI requires an organic compiler ecosystem that offers holistic solutions for a wide range of neural networks and are usable by non-FPGA experts. Once FPGAs are accessible to a growing community, their flexibility will mitigate performance bottlenecks, their low and predictable latency enables new edge applications, and their energy efficiency decreases the energy-footprint of AI applications. To achieve this goal, we propose and implement an organic compiler combined with the concept of Operation Set Architectures. This allows us to combine a large number of existing, but narrow, DNN-to-FPGA frameworks. Furthermore, the demonstrated one-click open-source organic compiler `DOSA` does not only increase the scope of potential solutions, it also increases the efficiency and is able to distribute large DNNs across many FPGAs automatically. Our results for a low latency DNN inference service on multiple FPGAs reveal a speedup of 18 and 84 times, and an energy efficiency-increase of 30 and 52 times compared to a GPU and CPU, respectively. We hope that our research showcases the advantage of combining specialized — yet restricted —- DNN-to-FPGA frameworks into one organic compiler, using the presented notion of Operation Set Architectures and Operation Set Generators and helps to lower the barrier for a wider deployment of FPGAs for AI.

ACKNOWLEDGMENTS

This work is partially funded by the EU Horizon 2020 Programme under grant agreement No 957269 (EVEREST). We would also like to thank our former colleague Mitra Purandare for her feedback in the early phase of the project and for picking the acronym DOSA.

NOTICES

‡ Intel, Intel logo, and Intel Xeon are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

** IBM and the IBM logo are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on ibm.com/trademark.

References

[1] J. L. Hennessy and D. A. Patterson, "A new golden age for computer architecture," *Communication of the ACM*, Jan. 2019. DOI: 10.1145/3282307.

[2] D. Amodei *et al.* (2018). "Ai and compute." https://blog.openai.com/aiand-compute, visited on 2022-05-19.

[3] A. Cohen *et al.*, "Inter-Disciplinary Research Challenges in Computer Systems for the 2020s," 2018.

[4] M. G. S. Murshed *et al.*, "Machine learning at the network edge: A survey," *ACM Comput. Surv.*, Oct. 2021. DOI: 10.1145/3469029.

[5] X. Wang *et al.*, "Convergence of edge computing and deep learning: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, Secondquarter 2020. DOI: 10.1109/COMST.2020.2970550.

[6] J. Fowers *et al.*, "A configurable cloud-Scale DNN processor for real-Time AI," *Proceedings - International Symposium on Computer Architecture*, 2018. DOI: 10.1109/ISCA.2018.00012.

[7] M. Blott *et al.*, "FINN-R: An End-to-End Deep-Learning Framework for Fast Exploration of Quantized Neural Networks," *ACM Trans. Reconfigurable Technol. Syst.*, Dec. 2018. DOI: 10.1145/3242897. arXiv: http://arxiv.org/abs/1809.04570v1 [cs.AR].

[8] Y. Umuroglu *et al.*, "LogicNets: Co-Designed Neural Networks and Circuits for Extreme-Throughput Applications," in *Proceedings of the 30th IEEE International Conference on Field-Programmable Logic and Applications (FPL)*, Gothenburg, Sweden: IEEE, 2020. DOI: 10.1109/FPL50879.2020.00055.

[9] Y. Umuroglu *et al.*, "FINN: A Framework for Fast, Scalable Binarized Neural Network Inference," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '17, Monterey, California, USA: Association for Computing Machinery, 2017. DOI: 10.1145/3020078.3021744. eprint: 1612.07119.

[10] S. Hooker, "The hardware lottery," *Commun. ACM*, Nov. 2021. DOI: 10.1145/3467017.

[11] B. Weiss *et al.*, "A power-efficient wireless sensor network for continuously monitoring seismic vibrations," in *2011 8th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks*, Jun. 2011. DOI: 10.1109/SAHCN.2011.5984921.

[12] S. Brigas *et al.*, "A Novel Design And Implementation Of A Wireless Sensor Network Aimed At Monitoring The Vibrations Produced By Oil
& Gas Activities," ser. Offshore Mediterranean Conference and Exhibition, OMC-2011-115, Mar. 2011. eprint: https://onepetro.org/OMCONF/proceedings-pdf/OMC11/All-OMC11/OMC-2011-115/1684977/omc-2011-115.pdf.

[13] J. Duarte *et al.*, "Fpga-accelerated machine learning inference as a service for particle physics computing," *Computing and Software for Big Science*, 2019.

[14] A. M. Deiana *et al.*, *Applications and techniques for fast machine learning in science*, 2021. arXiv: 2110.13041 [cs.LG].

[15] R. Herbst *et al.*, "Implementation of a framework for deploying ai inference engines in fpgas," in *Accelerating Science and Engineering Discoveries Through Integrated Research Infrastructure for Experiment, Big Data, Modeling and Simulation*, K. Doug *et al.*, Eds., Cham: Springer Nature Switzerland, 2022.

[16] T. Lieske *et al.*, "Dataflow optimization for programmable embedded image preprocessing accelerators," in *2016 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, Nov. 2016. DOI: 10.1109/ReConFig.2016.7857161.

[17] F. Jentzsch *et al.*, "Radioml meets finn: Enabling future rf applications with fpga streaming architectures," *IEEE Micro*, Nov. 2022. DOI: 10.1109/MM.2022.3202091.

[18] J. Duarte *et al.*, "Fast inference of deep neural networks in FPGAs for particle physics," *Journal of Instrumentation*, 2018. DOI: 10.1088/1748-0221/13/07/P07027. arXiv: 1804.06913.

[19] The Linux Foundation. (2022). "Open Neural Network Exchange (ONNX)." https://onnx.ai, visited on 2022-03-18.

[20] B. Ringlein *et al.*, "Advancing Compilation of DNNs for FPGAs using Operation Set Architectures," *IEEE Computer Architecture Letters*, Jan. 2023. DOI: 10.1109/LCA.2022.3227643.

[21] Fast Machine Learning Lab / hls4ml community. (2022). "hls4ml." https://fastmachinelearning.org/hls4ml/, visited on 2022-03-18.

[22] K. Abdelouahab *et al.*, "Tactics to directly map cnn graphs on embedded fpgas," *IEEE Embedded Systems Letters*, Dec. 2017. DOI: 10.1109/LES.2017.2743247. eprint: http://arxiv.org/abs/1712.04322v1.

[23] N. Tarafdar *et al.*, "Aigean: An open framework for deploying machine learning on heterogeneous clusters," *ACM Trans. Reconfigurable Technol. Syst.*, Dec. 2022. DOI: 10.1145/3482854.

[24] J. Ney *et al.*, "HALF: Holistic Auto Machine Learning for FPGAs," in *Proceedings of the 31st IEEE International Conference on Field-Programmable Logic and Applications (FPL)*, Virtual (Dresden, Germany): IEEE, 2021. DOI: 10.1109/FPL53798.2021.00069.

[25] A. Montgomerie-Corcoran, Z. Yu, and C.-S. Bouganis, "SAMO: Optimised Mapping of Convolutional Neural Networks to Streaming Architectures," in *Proceedings of the 32st IEEE International Conference on Field-Programmable Logic and Applications (FPL)*, Belfast, United Kingdom: IEEE, 2022. DOI: 10.1109/FPL57034.2022.00069.

[26] M. Blott, "Benchmarking neural networks on heterogeneous hardware," Ph.D. dissertation, Trinity College, 2021.

[27] T. Moreau *et al.*, "A hardware–software blueprint for flexible deep learning specialization," *IEEE Micro*, Sep. 2019. DOI: 10.1109/MM.2019.2928962. arXiv: http://arxiv.org/abs/1807.04188v3 [cs.LG].

[28] 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.*, Jun. 2018. DOI: 10.1145/3186332.

[29] K. Guo *et al.*, "[DL] A survey of fpga-based neural network inference accelerators," *ACM Trans. Reconfigurable Technol. Syst.*, 2019. DOI: 10.1145/3289185.

[30] M. Schneider *et al.*, "Ecba-mli: Edge computing benchmark architecture for machine learning inference," in *2022 IEEE International Conference on Edge Computing and Communications (EDGE)*, Jul. 2022. DOI: 10.1109/EDGE55608.2022.00016.

[31] J. Roesch *et al.*, "Relay: A new IR for machine learning frameworks," in *Proceedings of the 2Nd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, ser. MAPL 2018, Philadelphia, PA, USA: ACM, 2018. DOI: 10.1145/3211346.3211348.

[32] The ONNX community. (2022). "Open Neural Network Exchange Intermediate Representation (ONNX IR) Specification." https://github.com/onnx/onnx/blob/main/docs/IR.md, visited on 2022-03-18.

[33] The MLIR/LLVM community. (2022). "Multi-Level IR Compiler Framework." https://mlir.llvm.org, visited on 2022-05-31.

[34] C. Lattner *et al.*, "Mlir: Scaling compiler infrastructure for domain specific computation," in *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, Feb. 2021. DOI: 10.1109/CGO51591.2021.9370308.

[35] The MLIR/LLVM community. (2022). "The Torch-MLIR Project." https://github.com/llvm/torch-mlir, visited on 2022-12-21.

[36] ——, (2022). "Tensor Operator Set Architecture (TOSA) Dialect." https://mlir.llvm.org/docs/Dialects/TOSA/, visited on 2022-12-21.

[37] K. Majumder and U. Bondhugula, "Hir: An mlir-based intermediate representation for hardware accelerator description," *arXiv preprint arXiv:2103.00194*, 2021. DOI: 10.48550/ARXIV.2103.00194. eprint: https://arxiv.org/abs/2103.00194.

[38] W. Niu *et al.*, "Dnnfusion: Accelerating deep neural networks execution with advanced operator fusion," in *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, ser. PLDI 2021, Virtual, Canada: Association for Computing Machinery, 2021. DOI: 10.1145/3453483.3454083.

[39] R. Zhao and J. Cheng, "Phism: Polyhedral high-level synthesis in mlir — (work in progress)," *Workshop on Languages, Tools, and Techniques for Accelerator Design*, 2021.

[40] B. Ringlein, "Mapping of a Machine Learning Algorithm Representation to Distributed Disaggregated FPGAs," Ph.D. dissertation, Technische Fakultät der Friedrich-Alexander-Universität Erlangen-Nürnberg, Erlangen, 2022. DOI: 10.5281/zenodo.7957659.

[41] Xilinx Inc. (2016). "UltraScale FPGA Product Tables and Product Selection Guide (XMP102)." https://docs.xilinx.com/v/u/en-US/ultrascale-fpga-product-selection-guide retreived on 2022-05-31.

[42] E. Calore and S. F. Schifano, "Performance assessment of fpgas as hpc accelerators using the fpga empirical roofline," in *2021 31st International Conference on Field-Programmable Logic and Applications (FPL)*, Aug. 2021. DOI: 10.1109/FPL53798.2021.00022.

[43] M. Parker, "Understanding peak floating-point performance claims," Tech. Rep., 2014.

[44] A. Ivanov *et al.*, "Data movement is all you need: A case study on optimizing transformers," in *Proceedings of Machine Learning and Systems*, A. Smola, A. Dimakis, and I. Stoica, Eds., 2021.

[45] Xilinx Inc. (2022). "Performance and Resource Utilization for Adder/Subtracter v12.0." https://www.xilinx.com/htmldocs/ip_docs/pru_files/c-addsub.html#kintexu retreived on 2022-05-31.

[46] ——, (2022). "Performance and Resource Utilization for Multiplier v12.0." https://www.xilinx.com/htmldocs/ip_docs/pru_files/mult-gen.html#kintexu retreived on 2022-05-31.

[47] ——, (2022). "Performance and Resource Utilization for Floating-point v7.1." https://www.xilinx.com/htmldocs/ip_docs/pru_files/floating-point.html#kintexu retreived on 2022-05-31.

[48] B. Ringlein *et al.*, "System architecture for network-attached fpgas in the cloud using partial reconfiguration," in *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*, Barcelona, Spain: IEEE, 2019. DOI: 10.1109/FPL.2019.00054.

[49] B. Ringlein *et al.*, "A Case for Function-as-a-Service with Disaggregated FPGAs," in *Proceedings of the 2021 IEEE 14th International Conference on Cloud Computing (CLOUD 2021)*, Virtual Conference: IEEE, Sep. 2021. DOI: 10.1109/CLOUD53861.2021.00047.

[50] B. Ringlein *et al.*, "ZRLMPI: A Unified Programming Model for Reconfigurable Heterogeneous Computing Clusters," in *2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, Fayetteville, Arkansas: IEEE, May 2020. DOI: 10.1109/FCCM48280.2020.00051.

[51] ——, "Programming Reconfigurable Heterogeneous Computing Clusters Using MPI With Transpilation," in *2020 IEEE/ACM International Workshop on Heterogeneous High-performance Reconfigurable Computing (H2RC)*, IEEE, Nov. 2020. DOI: 10.1109/H2RC51942.2020.00006.

[52] D. Diamantopoulos *et al.*, "Phryctoria: A messaging system for transprecision opencapi-attached fpga accelerators," in *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2020. DOI: 10.1109/IPDPSW50202.2020.00023.

[53] A. Pappalardo. (2023). "Xilinx/brevitas." https://github.com/Xilinx/brevitas, visited on 2023-03-02.

[54] T. Alonso *et al.*, "Elastic-df: Scaling performance of dnn inference in fpga clouds through automatic partitioning," *ACM Trans. Reconfigurable Technol. Syst.*, Dec. 2021. DOI: 10.1145/3470567.

[55] Xilinx Inc. (2021). "Vitis AI User Documentation." https://www.xilinx.com/html_docs/vitis_ai/1_4/index.html, visited on 2021-07-04.

[56] A. Gholami *et al.*, "A survey of quantization methods for efficient neural network inference," Mar. 2021. arXiv: 2103.13630 [cs.CV].

[57] A. Pappalardo *et al.*, "Qonnx: Representing arbitrary-precision quantized neural networks," Jun. 2022. arXiv: 2206.07527 [cs.LG].

[58] PyTorch community, *Deep Learning with PyTorch: A 60 Minute Blitz — Training a Classifier*, https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html#training-on-gpu [Last accessed: October 19, 2022], 2022.

[59] F. Abel *et al.*, "An FPGA platform for hyperscalers," *Proceedings - 2017 IEEE 25th Annual Symposium on High-Performance Interconnects, HOTI 2017*, 2017. DOI: 10.1109/HOTI.2017.13.

[60] F. Abel *et al.* (2022). "The cloudFPGA Development Kit." https://github.com/cloudFPGA, visited on 2022-03-18.