Co-funded by the
Erasmus+ Programme
of the European Union

ERASMUS MUNDUS MASTER IN
IMAGE PROCESSING AND COMPUTER VISION

PÁZMÁNY PÉTER CATHOLIC UNIVERSITY
université de BORDEAUX
UAM Universidad Autónoma de Madrid

MSc THESIS

Efficient DNN Serving

Evaluating the feasibility of FPGAs for multi-tenant model serving

PRESENTED AT

PAZMANY PETER CATHOLIC UNIVERSITY

PÁZMÁNY PÉTER CATHOLIC UNIVERSITY

Author: CASPE, Franco Santiago

Academic Supervisor: NAGY, Zoltán
Industrial Consultant: MACE, Jonathan

DATE: June, 2021

Contents

Abstract

AI-powered services are deployed in a wide variety of online applications, requiring fast inference of deep neural network (DNN) models in order to ensure a good user experience. In this scenario, web applications define latency margins according to their needs, in a scheme known as inference as a service, where cloud providers aim to serve an arbitrary number of models with different service-level objectives (SLOs) with minimum footprint possible. In order to minimize latency and resource use, a novel cloud serving strategy called Clockwork leverages multi-tenancy of convolutional neural network (CNN) models while exploiting the predictability of their execution to order to schedule and execute inferences on GPUs, which has been proved to support thousands of models while meeting the required SLOs.

On the other hand, many cloud environments are featuring FPGAs as options for accelerating a variety of algorithms, since such a device allows the deployment of accelerator designs that are easy to maintain and can be designed to achieve high efficiency. The growing demand for computationally expensive, state-of-the-art DNNs, and the deployment cost of GPUs or other computing infrastructure, have motivated the design of a wide variety of DNN accelerators on FPGAs that feature different degrees of flexibility; the more rigid ones may allow executing a single DNN model, while the most flexible feature an instruction set architecture (ISA) that allows computing arbitrarily different workloads. Lamentably, for all design cases, the testing conditions are usually under isolated conditions, so for the high-performance model serving case that is studied for Clockwork, the feasibility of deploying efficient FPGA accelerators is uncertain.

Here, an analysis of existing CNN accelerators is performed for the datacenter context, from which an open-source ISA-based architecture called VTA is extracted and tested under realistic, Clockwork-like use conditions, considering not only throughput but also latency predictability under dynamically changing models. Next, the proposed accelerator is progressively modified in order to improve its latency distribution, demonstrating that predictability does not come for free when using FPGAs. Finally, it is empirically shown that ISA-based accelerators for current FPGA hardware do not provide energy benefits compared to other computing alternatives, which suggests that accelerators employing ISA are not suitable for the cloud acceleration paradigms

of interest. Less flexible designs might be able to provide acceptable performance, but only in cases where dominant models are present.

Acknowledgements

I would like to thank my laboratory supervisors, Dr. Jonathan Mace and Dr. Antoine Kaufmann, which gave me the possibility to conduct this work at the Max Planck Institute of Software Systems. I am very happy with this opportunity and I really appreciate the spirit of collaboration I sensed when working with you, as well as the support and ideas you have given me for this work. A highlight of my experience has been undoubtedly our discussions and brainstorming sessions, where I got to recognize the beauty and also the challenges of conducting research and approaching problems from a high-level point of view.

I would also like to thank my University supervisor, Dr. Nagy Zoltán, for introducing me to the topics of image processing and neural networks acceleration on FPGAs and for his patience and thorough support on the different projects I conducted during my Master's, including this Thesis, as well as the freedom and trust given for me to explore and discover the field while working. It has been a very enjoyable learning experience.

Finally, I would like to thank my colleagues and friends, Anne-Claire, Jingwen, Lukas, Nano, Talal, Julia, Berta, and Marta, for the excellent time we spent together, studying, working, and traveling. Thank you.

Chapter 1

Introduction

1.1. Background

Machine learning (ML) model inferences are widely employed in web applications, often taking place on the critical path of web applications [1], [2], where they are handled by model serving services [3]–[6], distributed systems that support a wide variety of pre-trained ML models at low latency. From the service provider point of view, there exist additional challenges for meeting such requirements at scale; keeping device use and energy efficiency high are key to maximize throughput and minimize costs [7].

When designing a serving application, latency and throughput requirements are usually achieved by implementing specific strategies that orchestrate and manage requests in the most effective way possible, namely by using a scheduler that dispatches tasks across a number of distributed worker machines. Recent designs for neural network inference serving aim to bound the requests latency and maximize worker use [6], [8] by employing multi-tenancy of models and exploiting determinism on their execution time. Bounding latency guarantees that a produced result is useful to the client while limiting the number of workers is especially important when considering that for the majority of cases, deployment does not take place in a dedicated serving infrastructure, but in virtualized cloud environments [9]–[11], where application footprint, that is, the number of instances used for the application, is directly proportional to the rent cost of the service.

One of the most important sources of operative cost in a server infrastructure comes from energy consumption [12]. Many efforts in the design of specialized ML hardware are being conducted in industry and academia [7], aiming to execute workloads in a more efficient manner than canonical computing infrastructure such as CPUs or GPUs [13]. For instance, reconfigurable devices such as Field Programmable Gate Arrays (FPGAs), allow for the deployment of efficient,

specialized hardware while also being highly compatible with the virtualized paradigm of a cloud environment [11], [14]. Distributed applications leveraging FPGAs are easily (un)scalable, with working machines being configured for potentially different tasks as they are taken or freed by the application of interest. Furthermore, such devices avoid the barriers of entry of application-specific hardware (ASICs), for which the cost of their design, testing and deployment phases may be only justifiable for very high scale applications [15] [16].

Considering this scenario, we are interested on the possibility of combining a resource-sharing approach with the use of FPGAs, deploying specialized accelerators for virtualized environments, in order to jointly leverage the economic benefits offered by both approaches. This strategy nonetheless, entails challenges related to the production environment on which such programmable devices are typically deployed: a ML inference resource sharing system requires workers to support multiple models, but current FPGA design approaches typically leverage the benefits from performance and efficiency by exploiting the use of highly specialized accelerators, that often support single workloads [17] and would require a slow device reconfiguration operation for the worker to switch tasks. We assume that such a delay [18] is not compatible with the resource sharing scheme of interest. Thus, an FPGA accelerator with a certain degree of built-in flexibility is required. Since this condition may involve a reduction in efficiency and/or throughput, the objective of this work is to empirically quantify that trade-off for current FPGA accelerator designs, in order to determine the feasibility of the whole approach.

In this work, an open-source FPGA DNN accelerator based on a flexible instruction set architecture (ISA) is deployed and tested under a simulated resource sharing scheme obtaining performance, latency distribution, and power consumption measures. Then, the design is modified in order to improve its latency characteristics. Finally, a comparison with current datacenter GPU acceleration technology is established, concluding that ISA-based accelerators are too general to exhibit energy efficiency gains, being unsuitable for the application of interest.

1.2. Outline

The summary of the chapters of the thesis work:

Chapter 2 This chapter sets the background for this thesis, describing related work in multi-tenancy serving and hardware accelerators for DNN inference.

Chapter 3 This chapter presents our motivations, hypothesis, and challenges to multi-tenancy model serving with FPGA accelerators. Next, we evaluate our options and describe the technical

background related to our DNN accelerator reference design. Finally, we present some caveats of the design and propose improvements to enable implementation for our use case.

Chapter 4 In this chapter, we assess the predictability and efficiency of our reference design, the impact of our architecture modifications, and we compare the power rating of our accelerator with that of a state-of-the-art GPU. We present our experimental setup and the results we obtained, to finally draw conclusions related to the feasibility of multi-tenancy implementation of ISA-based accelerators.

Chapter 5 This chapter presents a summary of the problems solved compared to the objectives presented in the introduction and in the Thesis Proposal Form. We then point out opportunities to move forward, motivating future work.

Chapter 2

Related Works

In this section, we present and analyze published work related to resource-sharing systems and DNN inference infrastructure.

2.1. State of machine learning

Current machine learning algorithms, especially deep neural networks (DNN) have led to breakthroughs on accuracy rates for text, speech, time series, and image processing, being deployed for an increasingly variety of applications, to name a few, computer vision [19] [20], financial forecasting [21], ad-targeting [22] [23] and virtual assistants [24] [25]. DNNs are composed of multiple layers of artificial neurons tuned through non-linear inner products and pooling operations [26].

The inherent computational complexity of DNNs their vast field of application present challenges that have sparked considerable interest in making training and inference easier, faster, and more efficient. For instance, training and inference can be accelerated by exploiting the inherent parallelism of DNN models. This can be tackled by the development of hardware including ASICs [27][15], FPGA designs [17] [28], and GPUs [29]. Moreover, the heterogeneity of the training frameworks [30] and inference infrastructure has called for model exchange formats [31] [32] and optimizing graph compilers [33][34] that allow port multiple models to different computing architectures, unlocking their particular advantages.

2.2. Low-latency DNN model serving

DNN models are being increasingly deployed on the critical path of web applications [35].

Multi-tenancy To ensure prompt execution, inference has been decoupled from other tasks, now being conducted by specialized data center infrastructures [36][3][4]. These distributed

systems avoid the deployment of separate computing hardware for each application, implementing a resource-sharing strategy among different customers, denoted *multi tenancy*[37], which shrinks application footprint while keeping latency at margin. In this case, the system stores a set of models for each client who, in time, submit appropriate inference requests. A model serving back-end manages the user's models and computing resources accordingly, loading the model in a worker (if not loaded previously), performing the inference, and duly returning the result.

Unpredictable Workloads Model serving shares similar concerns with other specialized data center services, such as streaming or storage services [38]. They manage workloads of different users and balance load across multiple workers and hardware accelerators.

For the study case of interest, the deployment of a high-performance resource-sharing system presents several challenges. Typically a large number of models are being handled in one single system. They cannot be present at the same time in all workers due to limited RAM. Considering this, request fluctuations may incur in models being loaded and unloaded from the workers very often, reducing throughput. It is worth noting that such fluctuations are not deterministic; it is assumed that they are unpredictable [39].

Service-Level Objectives To achieve interactive speeds, model serving services operate under real-time constraints; requests that are not attended on time are considered useless to the client. Most cloud and data center services have *service-level objectives* (SLOs) that indicate the performance that clients can expect from the service [40]. Typically, a SLO in terms of latency is enforced, in the order of milliseconds [41] [42] [43]. For example, it might specify a 10ms average response time or a 40ms 99^{th} percentile response time.

To manage the client's requests, existing model serving systems feature a scheduler that dispatches tasks to workers and retrieves results. Such a system selects which models to be executed according to the task's characteristics and worker's state variables.

Existing DNN Serving Systems In Clipper [3], authors propose a resource-sharing system that caches inferences on a per-model basis, implementing adaptive batching to maximize throughput given a latency target. Moreover, to improve latency, a model selection strategy adopts a straggler mitigation technique to render predictions without waiting for slow models. Finally, models are provided in TensorFlow format and executed using its native GPU backend. In Infaas [6], the scheduler receives another degree of freedom by introducing model-vertical autoscaling that, through model selection, upgrades or downgrades an inference request to a differently optimized model variant. The model variant concept comprises different DNN versions

inside a single model family (i.e. Resnet18, Resnet50, Resnet152), with each member optimized for different batch sizes using a graph compiler.

Previously presented designs used separate containers/virtual machines for each model. Current system design approaches are trending towards more tightly integrated systems, where the scheduler puts all models in a unified process that controls not only worker allocation but also the execution. This allows it to swap models at fine time scales in the order of milliseconds. Clockwork [8], leverages a modified GPU runtime, to achieve predictable GPU workers. This allows to exploit the deterministic nature of DNNs. Since an inference process takes no conditional branching and involves the execution of a fixed number of operations, latency on such GPUs becomes deterministic. Furthermore, by consolidating the decision taking in a centralized controller, they are able to predict the turnaround time of a request, making better-informed scheduling decisions, and achieving higher resource use and SLO compliance than previous approaches.

2.3. DNN models and computing requirements

There are several DNN types, each one with different target applications. In this section we present an overview of the most common, explaining how they vary in their computational characteristics. This is important for a model serving system because ideally, it should support all of them.

DNNs can be classified according to the way in which inner products are conducted in the layers. This yields different categories or types [44], each one better suited for a specific set of tasks. At the same time, the different computation characteristics of the models present varying challenges to an efficient implementation.

In *Convolutional Neural Networks* (CNNs), best suited for classification and generation of images and audio, each layer is a set of nonlinear functions of weighted sums of nearby subsets of outputs from the prior layer. Moreover, the convolution is performed in a window that is slid across the input; this minimizes layers' memory footprint, as inputs and weights are heavily re-utilized.

In *Multi Layer Perceptrons* (MLPs), used as classifiers and recommender systems, each new layer is a set of nonlinear functions of the weighted sum of all outputs from a prior one. As each neuron holds one weight for every single input, there is no re-use in this model, which incurs into higher memory use and access rate when executed.

Finally, *Recurrent Neural Networks* (RNNs), which are used to process time series data such as speech and text, present layers where each neuron has not only a learned weight but also a

state value that is updated across the execution, requiring a higher frequency of memory access for each neuron output generated.

It is possible to infer that, although all models are based on inner-product layers and non-linear operations, the actual computing requirements are not similar. When high throughput is required, computing hardware has to prioritize different aspects of the datapath, as the bottlenecks may be different for each type of DNN. RNNs will require a large memory cache to quickly access inputs, weights, and states when performing inference, whereas CNNs are usually bounded by computing power, requiring more computing resources, but less memory resources than the other accelerators. MLPs are on a middle ground, but in highly parallel architectures, their throughput is more likely to be memory-bounded.

We employ the *operational intensity* [45] definition of an algorithm to characterize the different kinds of DNN layers. For similar input and output data sizes, CNNs are more intense that MLPs, which in turn, are more intense than RNNs [46] [15].

2.4. DNN inference hardware

The end of Dennard scaling [47] and diminishing benefits from transistor scaling [48] have propelled an era of Domain Specific architectures. As such, system designers employ accelerators to enable performance improvements necessary for emerging workloads [49]. Cloud-backed inference is currently enabled by various forms of custom devices, such as GPUs [29], ASICs such as Google's TPU [15], and FPGAs, like in the Microsoft Brainwave Project [28]. When comparing with a general-purpose computing engine, like a CPU, and for a specific kind of DNN, each one of these architectures presents specific advantages, which will be described in the next subsections.

2.4.1 GPUs

Current datacenter GPU cards [50], are composed of many small computing cores that can operate in parallel, several caches, and a DRAM unit. Each of those cores is saturated with Arithmetic Logic Units (ALUs) that operate conforming a compute scheme known as Single Instruction, Multiple Thread (SIMT), i.e. all units receive the same instruction, following the same program counter. This proves extremely efficient when computing highly parallel, compute-bound workloads. As the computation time surpasses that of the communication to DRAM, both operations can be performed concurrently with the system not suffering from data starvation. Such a technique is called *latency hiding* and allows to maximize hardware use for a specific workload. This nonetheless cannot ensure full utilization of the system. It is common that workloads occupy just a part of the ALUs of a specific thread, with the rest of them executing

operations for which their results are unused. When executing DNNs, a technique known as *batching* increases the computational intensity of the algorithm, by performing inference of many inputs at the same time. Since the weights of a specific model are always the same regardless of the input provided, batching allows the system to fetch weights only one time for each batch, reducing data movement, and increasing efficiency. Nevertheless, in a resource-sharing scheme, this technique can only be applied if more than one request for the same model is received. Since workload arrivals are unpredictable, there is no guarantee that batching can be exploited on the majority of the cases.

At the datacenter, resource-sharing strategies employ batching to increase device use and maximize throughput, trying to delay similar requests in a queue until a considerable batch is formed. Nevertheless, the strategy is good as long as none of these inferences are executed outside their valid time frame.

GPUs show high performance when computing CNN and MLP inferences, even with a small batch size, as the algorithms are highly parallel and weight or input reuse strategies can be applied. For RNNs, GPUs do not show good efficiency for small batches, as the low amount of parallelism and high memory use exhibited by such algorithms is not enough to fully occupy the computing cores, with CPUs in many cases being more suitable for the task [51].

Since GPUs are suitable for high throughput inference, they are used at the cloud and can be rented to deploy web applications that involve DNN inference. [10] [9]

2.4.2 ASICs

Since DNN serving systems are specialized systems, a possible approach is to use specialized hardware to process requests. Recent Application-Specific Integrated Circuits (ASICs) for DNN inference aim for reducing data movement, and the energy cost it entails, hence increasing performance. For example, designs presented in Yu-Hsin Chen et al. [27] and Zidong Du et al. [52] exploit the two-dimensional data distribution of CNNs to generate single computing cores for spatial processing, maximizing convolution efficiency.

Next, Google's TPU [15] takes a more general approach. It accelerates matrix multiplications by tiling or splitting them into smaller operations that can fit a systolic array. The systolic array can compute a matrix-vector multiplication of 256 elements in one clock cycle. The system is finished with a big scratch register memory, an Instruction Set Architecture (ISA) and an optimizing compiler that allows describing CNN, MLP, and RNN workloads in terms of such instructions, maximizing performance and minimizing data movement. TPU workers in Google's datacenter feature fast DRAM and PCI Express interfaces, showing better performance in all of

the three kinds of DNNs when comparing with state-of-the-art GPUs [53]. As each workload is executed sequentially by a single computing engine, batching does not improve efficiency, except in very specific cases where the systolic array can be occupied by more than one input throughout the whole computation. Thus, maximum throughput is achieved with minimum latency.

From a management standpoint, implementing ASICs in a datacenter presents a big barrier to entry, since the design and deployment costs have to be taken into account, so the volume of inferences need to justify such an investment. Moreover, unlike GPUs, ASICs, including TPUs, can only be efficiently used for DNN inference, being unable to perform in other applications traditionally targeted for datacenters, like video encoding, regression tree inference [54], or other high-performance computing tasks not based on neural networks.

2.4.3 FPGAs

Device Descritpion An FPGA is an integrated circuit composed of logic and memory units called *slices* and *Block RAMs (BRAMs)* for which its interconnection can be modified as many times as needed by updating the connection map on-chip, which is called *bitstream*. In a nutshell, different connections yield different functionalities which are actual hardware implementations. FPGAs can be codified either by manually describing the connections (Register Transfer Logic), by describing circuits and processes (Hardware Description Language (HDL)), or more recently by describing a sequential program for which its functionality can be re-interpreted as a combination of circuits and processes (High-Level Synthesis (HLS))[55].

FPGAs at the datacenter FPGAs have found widespread deployment at the datacenter [14] [11] because they comply very well with the idea of a virtualized environment: in this case, even the hardware design can be virtualized, allowing simple horizontal replication of applications, which are configured in both software and hardware for a specific task. Moreover, when aiming for efficient deployment of a specific algorithm, FPGAs benefit from specialization and shorter-than-ASIC design stages.

Interfacing software with reconfigurable hardware is one of the biggest challenges when deploying solutions on FPGAs. At the datacenter, two major alternatives have been developed, namely the use of FPGA boards connected to a CPU host over a PCI express interface (PCIe), and the use of an FPGA System on Chip (SoC) [56]. PCIe boards feature an FPGA devices, and a local DRAM card. The Host CPU can write to the DRAM and command the device to process the data stored there. A simple communication scheme between a Host CPU and an FPGA board is shown in Figure 2.1. The SoC is composed of a Processor System (PS), which is a standard CPU, typically an ARM Cortex core, etched on the chip substrate, and a sepa-

rated reconfigurable logic section called Programmable Logic (PL). Both devices share the same DRAM, which can be read and written directly by either one. SoCs leverage the use of complex operative systems, like Linux, to provide a standardized point of access to the hardware inferred on the PL. Moreover, they can be developed standing alone, without the need for a standard CPU connected through PCIe. The basic communication scheme of a SoC is shown in 2.2.

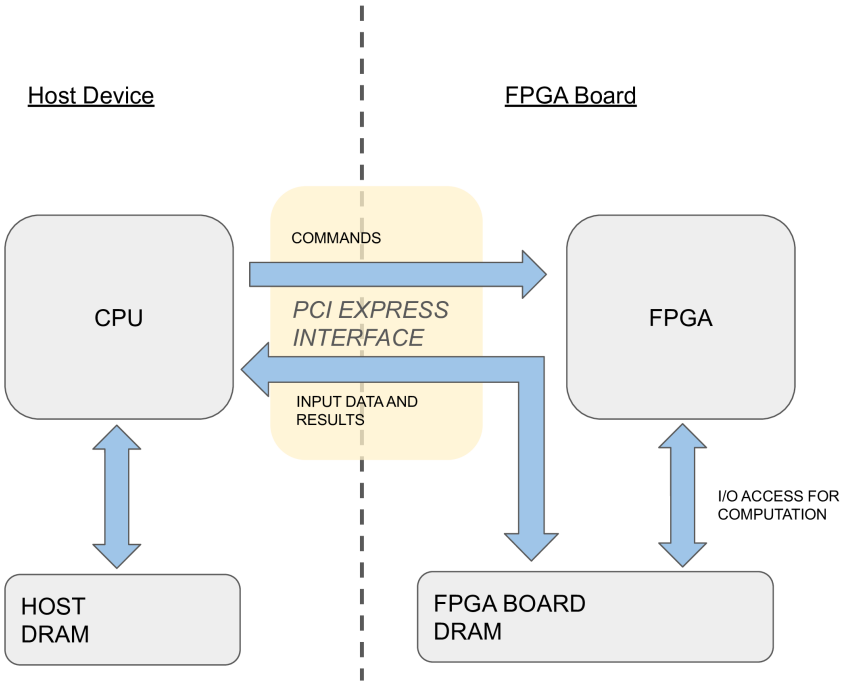

Figure 2.1: Communication scheme between a host CPU connected to a PCI Express FPGA board.

ML inference on FPGAs Considering ML workloads at the datacenter, although FPGAs have proven good for high volume inferences [57], [58], they present challenges for multi-tenancy implementation. Switching functionality on an FPGA is done through a reconfiguration process that might take tens of seconds to finish. Even though partial reconfiguration can help reduce these waiting times to the order of seconds [59], such a delay is still unacceptable in resource-sharing systems.

By configuring FPGA workers with reusable, flexible, and shareable accelerator designs, authors in the Microsoft Brainwave and VineTalk papers [28], [60], were able to implement a resource-sharing system for DNN inference that does not rely on reconfiguration. The first work presents a specific case of serving a subset of RNNs and the second describes a proof of concept for model serving targeting financial applications, showing that FPGAs are effective in datacenter applications to serve long-lived tasks or a subset of DNN models. In light of these results, we believe that FPGA accelerators with fast task switching times may also be a good idea for our

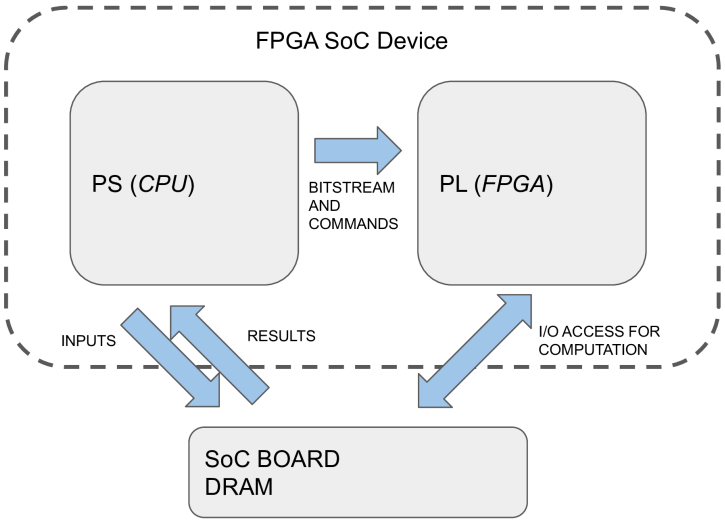

Figure 2.2: Communication scheme on an FPGA System-on-Chip (SoC)

multi-tenant DNN serving application of interest.

2.5. DNN inference on FPGAs

In this thesis, our focus will be on DNN inference on FPGAs. In this section, we discuss current efforts for efficient DNN inference on FPGAs, and categorize them from the feasibility standpoint of a multi-tenancy, resource-sharing system approach.

FPGA DNN Accelerators A block schematic of a typical FPGA-based neural network accelerator system is shown in Figure 2.3. The host CPU issues workloads and commands to the accelerator and monitors its status. On the FPGA logic part, a controller is implemented to communicate with the host and to generate control signals to all the other modules on FPGA. The on-the-fly logic part is implemented if data loaded from external memory needs preprocessing. The computation units are usually arranged in a spatial configuration that can concurrently compute one or more loops involved in convolution or matrix multiplication operations. Finally, the on-chip memory resources of an FPGA chip, called Block RAMs (BRAM), are typically too limited compared with the large DNN models. So for common designs, a two-level memory hierarchy is established, combining a dedicated DRAM and on-chip buffers.

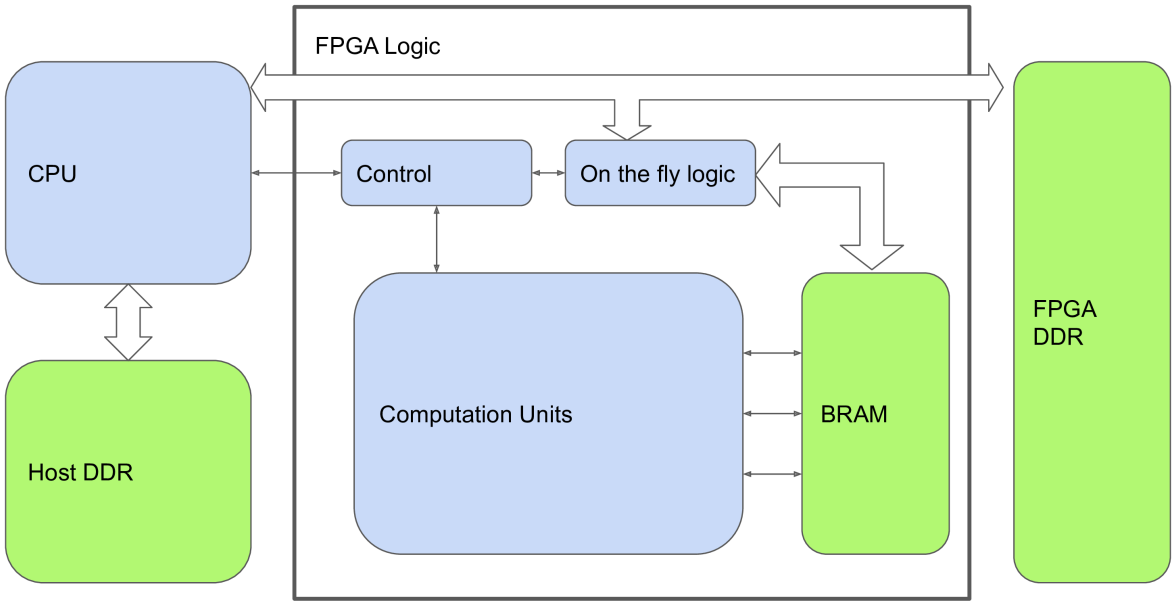

Figure 2.3: Block graph of a typical DNN FPGA accelerator. Adapted from [17]

Accelerator principal characteristics A resource-sharing scheme requires accelerators that are fast, flexible and energy-efficient. This section presents several DNN accelerators implemented on FPGAs, which are analyzed from multiple points of view. The key dimensions we care about are:

- **DNN Layer type and sizes:** Refers to the DNN layer (Convolutional, Fully-Connected, Recursive, Pooling) and the sizes that can be executed on the accelerator.
- **Performance:** Qualitative and quantitative analysis of latency and throughput achievable.
- **Predictability:** Ability to confidently determine inference throughput and latency within certain margins.
- **Energy efficiency:** Refers to the number of operations that are carried out per energy unit, or throughput achieved per Watt.
- **Flexibility:** Ability to switch between DNN models implemented on the same accelerator.

2.5.1 Accelerator design approaches

A key design decision on any FPGA accelerator is related to the way in which the computing and memory resources are mapped to the computing primitives that have to be performed. It is natural to assume that different mapping strategies incur in different trade-offs of the dimensions mentioned before.

Computation and Communication The core operation of DNN inference is the Multiply-and-Accumulate (MAC) operation, which provides the base for matrix multiplication. FPGA accelerators usually perform inference in fixed-point arithmetic, as such operations are efficiently mapped to computing resources present in current FPGAs. State-of-the-art devices feature computing units called DSPs, distributed across the reconfigurable logic that typically, and through vendors, can perform at least one 18×27 bit MAC operation in one clock cycle [61] [62], with common design frequencies in the order of 300 MHz. Current top-of-the-line devices feature DSPs in the order of thousands, thus, a device with 3000 DSPs units, could theoretically achieve a throughput of about 0.9 trillions of MACs per second, or 1.8 Tera Operations per Second (TOPS), being able, in theory, to execute all common CNNs in the order of tens of milliseconds [63]. Throughput can be further enhanced by employing fine-grained hardware design strategies such as DSP Double Data Rate [64] and efficient mapping of MAC operations [65].

Another characteristic that defines throughput and efficiency in a design is the nature of the data movement [66]. Fetching data from DRAM is expensive in terms of time and energy, and can lead to data starvation of the computing cores, reducing throughput and efficiency. Then, FPGA memory resources, which are relatively scarce, are mapped into internal buffers located nearby the compute units to minimize movement by exploiting data reuse as much as possible.

Template-based accelerators The design spectrum of DNN accelerators can be described in terms of the primitive operations carried out by the computing cores and the data movement they generate. One end of it responds to traditional algorithm deployment techniques on FPGAs: provided enough resources are available, it is possible to map the entire algorithm to the device, i.e. implement all DNN layers with their specific sizes on-chip, forming a pipeline that, once filled, can generate predictions at a throughput defined by the slowest link of the chain [67]–[69]. Here, data movement is minimized: the algorithm is described in a dataflow architecture with partial results traversing through the physical layers. Weights can be either fetched from DRAM or buffers local to each layer. This approach, which we denote *Template-based*, allows to fine-tune an accelerator for a specific DNN architecture, achieving high throughput and efficiency. It cannot, however, execute any other type of DNN architecture without re-configuring the FPGA, which incurs high waiting times to exchange models.

ISA-based accelerators The other end of the spectrum is represented by accelerators that implement some DNN fundamental operations in parallel such as matrix multiplication, ReLU or pooling, on unified computing cores, trying to exploit as many resources as possible [70] [64]. Here, DNN inferences are described in terms of the available primitives, by relying on

an Instruction Set Architecture (ISA) that is implemented in the design. We identify such accelerators as *ISA-based* designs. In terms of flexibility, they can execute arbitrarily different workloads (within the limits of their ISA), just by changing the program in execution, operation that can be done quickly. Data movement, nevertheless, becomes much higher, since they have to fetch instructions, weights, and inputs from DRAM to perform the operations, while also storing intermediate results either to a local buffer or back to DRAM.

Middle-ground: recurrent accelerators Finally, in the middle, a design strategy may reduce data movement with respect of ISA-based architectures, by implementing one or more recurrent and configurable computing cores, for example, convolution units [71] [72]. This approach reduces the instruction set to simpler configuration fields but limits the type of operations that can be carried out with sensible efficiency since potentially different workloads from a DNN model share one or a few cores.

2.5.2 Review of representative designs

Binary Neural Networks A template-based design of high throughput and efficiency was presented in [69]. The work presents a framework that is used to map CNNs and MLPs trained as described in [73]. The obtained accelerators can perform inference under reduced arithmetic precision, even with 1-bit weights and inputs. 1-bit MACs can be mapped very efficiently to logic slices on the FPGA, without the need of DSPs, increasing the theoretical peak performance of the device. Using the framework, each layer of a model is mapped onto a processing engine in the FPGA, forming a pipeline. The framework allows targetting more computing resources to bigger layers, balancing the footprint of the accelerator in the device with the maximum throughput achievable, in a process known as *design space exploration*. The totality of weights is stored on-chip, as their footprint is very small, so no DRAM access is needed while performing inference, which generates a completely isolated computing engine that is very predictable, as it does not need to interact with any other device while running.

As with any template-based model, this approach remains limited to certain DNN models that feature a low count of layer levels. Since every layer is implemented on-chip, computing resources are depleted very quickly, only being able to implement relatively small models. Next, although this design represents a notable example of a template-based accelerator, 1 or 2-bit inferences do come with a high reduction in accuracy which may be unacceptable for most applications. Finally, in terms of flexibility, it is not possible to switch between models without reprogramming the device, which is detrimental for a resource-sharing system.

DNN Builder Another template-based accelerator framework is presented in X. Zhang et al. [68] for classification of HD images. In this work, a software utility allows to parse and transform 16-bit or 8-bit quantized Caffe [30] models, perform design space exploration and generate a dataflow architecture with up to 16 layers. Weights and intermediate feature maps are fetch from DRAM using a column-based cache scheme that minimizes data movement. This approach does not suffer from high accuracy degradation, being up to 4 times more efficient than GPUs and presenting comparable throughput at the time of publishing. Next, latency can be inferred from the design, as it follows a dataflow paradigm. As a template-based accelerator, it requires device reconfiguration to switch between models, slow for resource-sharing systems.

Cloud DNN A hybrid, template-based with recurrent layers was presented in Y. Chen et al. [71]. Here, a Caffe model is parsed and a design space exploration strategy is performed to maximize throughput. The goal is to generate an accelerator composed of three groups of recurrent layers. Each group is mapped to a Super Logic Region (SLR) [74] of a state-of-the-art FPGA. The software generation instance estimates the clock cycles taken by each accelerator, greedily finding the best configuration for a specific model. Many layers of the DNN model can be computed by the same accelerator. Each group can have up to three configurable computing engines that perform convolution, pooling, and matrix multiplication. Weights and parameters for each workload are fetched from DRAM. Intermediate feature maps are stored in buffers located between the accelerators. At any time, the intermediate feature maps can be processed either by the previous accelerator or the next one, hence, data needs to traverse all computing cores to be written back to DRAM. This minimizes data movement, and allows for fast execution, with efficiency margins comparable with GPUs.

Although template-based, since each computing core can execute multiple workloads, it is possible to assume that an accelerator tuned for a specific DNN model can execute others. Nevertheless, it is important to have into consideration that each computing core is adjusted for a specific set of DNN workloads, allocating a specific number of DSPs in a specific spatial configuration of feature map size and input channels. Workloads that present highly different configurations may present very low efficiency, with the computing core executing many empty operations. Throughput and efficiency characteristics of this design have been further improved in [67].

Xilinx DPU An ISA-based set of designs has been developed by Xilinx, targetting DNN inference on multiple devices, and being integrated with a DNN deployment framework called Vitis-AI [75]. This set of processors, called DPUs, come in different sizes and with different

memory interfaces, to adapt to specific requirements that pose each FPGA device [64]. Although their architecture is closed, they are based on one or more, 8-bit matrix multiplication computing engines that can perform convolutions and inner products in a tiled form, across feature map and input channel dimensions. *Tiling* refers to the process of dividing a workload into separate pieces that can be computed using non-divisible instructions. DPUs are quick to switch between models because they don't rely on device reconfiguration. By loading a new program, weights, and input data into DRAM, the architecture is ready to perform a new inference, useful for resource-sharing systems [60]. Next, although there exist latency-optimized versions of DPU designs, no information regarding latency predictability has been published. Finally, even though the biggest DPUs feature throughput for some models that is comparable to that of state-of-the-art GPUs, Xilinx does not disclose any information about their efficiency. The lack of power information and closed architecture, which cannot be revised and which can only target very specific FPGA PCIe cards, could be discouraging for an extensive deployment in datacenters.

Microsoft Brainwave The Microsoft Brainwave [28] project aims to serve a very specific subset of five RNNs that are massively employed by some Microsoft products. The accelerators feature a ISA-based architecture and a compiler to generate the workloads for inference in a custom floating-point format. Since RNNs have a big memory footprint but do not require much computing power, the design challenge relies on fully mapping the accelerator buffers into the memory resources of the FPGA to avoid data starvation. So, an accelerator is distributed across multiple FPGA devices maximizing memory access of the computing cores. These accelerators are flexible, fast and exhibit low latency, but can only provide good throughput for the specific RNN cases.

VTA Finally, in [70], the authors present an ISA-based accelerator called VTA, that aims to leverage the TVM [33] optimization capabilities to accelerate DNNs in SoC FPGA devices. The architecture is programmable, designed to be extensible in the face of evolving workloads, and can quickly switch between models. VTA achieves this flexibility via a parameterizable matrix multiplication computing core, which can be expanded to exploit resources available on the device, a two-level ISA, and a Just In Time (JIT) [76] compiler that allows the operations to be computed either in the PS or the PL. The two-level ISA implements concurrent computing, memory tasks, and a wide variety of operators with single-cycle tensor-tensor operations, allowing to perform latency hiding. The run-time system, based on the JIT compiler allows flexible code-generation and heterogeneous execution that enables effective use of the architecture.

This accelerator can be deployed on different FPGA vendors and is flexible in its hardware

configuration, which makes it scalable to differently sized FPGAs to target the desired throughput. It also features an open architecture with high compatibility for current workloads through TVM and a dedicated compiler. The authors claim it features better efficiency than a CPU, but do not give specific power measures. Since VTA is a single-core architecture, latency can be directly related to throughput, with authors presenting the obtained average frame rate. No information has been published regarding predictability. Finally, a recent work for which its source code is yet to be released aims to increase the throughput of the architecture, by deploying multiple computing cores in the accelerator, enabling multi-threaded execution [77].

Chapter 3

Methods

This chapter presents our motivations, hypothesis, and challenges to multi-tenancy model serving with FPGA accelerators. Next, we evaluate our options and describe the technical background related to our reference design. Finally, we present some caveats of the design and propose improvements to enable implementation for our use case.

3.1. Problem statement

3.1.1 Motivation

Due to their flexibility, FPGAs at the datacenter can accelerate a vast array of different applications for different clients without the need to acquire expensive and specific hardware. This includes DNN inference tasks, with some designs featuring quick task switching (ISA-based accelerators). On the other hand, a managed multi-tenancy inference system can help minimize the footprint of a cloud application by maximizing workers' use while complying with the requested SLOs. We are interested in assessing the feasibility of a DNN inference system that would benefit from both approaches, to ultimately design a DNN serving application that presents low economical barriers to entry, and is scalable, efficient, and effective in terms of SLOs.

3.1.2 Challenges

Unknown feasibility Based on the review presented in the previous chapter, we see that a DNN inference system based on multi-tenancy on FPGA accelerators has not been explored before. Existing works on FPGAs for DNN inference typically evaluate the accelerators under isolated conditions, but datacenter application of interest needs to continuously swap and execute different models in the workers. We do not know if it is feasible to leverage the benefits from multi-tenancy and FPGA acceleration at the same time.

Non-trivial selection We understand that the requirements enforced by the resource sharing system condition the selection of the accelerator. The considerable diversity of models that have to be supported may limit the benefits that can be obtained from accelerator specialization since we do not want to incur in device reconfiguration to switch tasks. Across a large selection of accelerator designs, we seek an efficient design that is able to support multiple models, a feature that is a requirement for workers deployed in a resource-sharing inference system.

Accelerator benchmarks are simplistic Current resource-sharing systems employ tightly integrated schedulers that rely on workers' performance predictability to dispatch tasks. Nevertheless, the accelerators presented in the literature review are bench-marked with simplistic tests that only quantify the average throughput for specific DNN models. We are interested in estimating a latency distribution of the FPGA workers to assess whether they are predictable or not.

Finally, we have observed that on many works, there is a tendency to highlight throughput without having into consideration the energy burden it entails. Although highly parallel architectures such as DNN accelerators tend to exhibit in all cases better performance than a CPU, only a few designs do actually present an objective efficiency measurement to compare against other datacenter alternatives such as GPUs.

3.1.3 Hypothesis

As stated before, ML inference serving environments have successfully exploited the quick task switching capabilities of FPGA accelerators in order to implement successful cloud serving applications [28] [60], serving a limited selection of models. We believe that this approach can be extended to a multi-tenant environment that serves a wide variety of DNN models, as long as the workers exhibit performance predictability, and provide efficiency margins comparable with other inference architectures.

3.1.4 Goal for this work

By observing the key dimensions of accelerator analysis presented in Section 2.5, we decide to prioritize flexibility over performance or efficiency because multi-tenancy cannot be exploited at task switching rates in the order of seconds, so we focus on ISA-based architectures. Moreover, we have seen that ISA-based architectures can exhibit throughput that is comparable with other computing architectures [28], [64]. For the study case, we choose CNN and MLP inference because as of today, such accelerators support the widest variety of models.

In this work, we analyze if ISA-based accelerators can comply with the predictability and efficiency requirements of multi-tenancy systems in order to assess if the combining approach is viable or not. We choose an accelerator reference design and generate tests that realistically recreate the working conditions of a multi-tenancy system and evaluate how the requirements affect the performance and predictability. Then, we enhance the architecture of our design of reference by considering the use case of interest. Finally, we empirically draw our conclusions from the performance and efficiency measurements that are conducted under different working conditions.

3.2. ISA-based accelerator

3.2.1 Evaluation

Available options Presented in the previous chapter, there exist to our knowledge, only two ISA-based architectures openly available to deploy DNN accelerators, namely Xilinx's DPU [64] and VTA [70]. The latter is especially interesting because it features an end-to-end open stack for inference, based on TVM [33]. Furthermore, unlike the DPU designs, VTA's architecture is open-source and can be modified and implemented on multiple FPGA vendors. Finally, it is worth mention that the hardware architectures of both DPU and VTA are parameterizable. This allows adjusting the accelerator's performance capabilities according to the task's requirements and the device's available resources where they are deployed.

Selection We select VTA's architecture because it is the only architecture that is open-source, and fully described; the only one that may allow us to extract informed conclusions from our experiments. Unlike VTA, DPU's available architectures are offered as black-boxes, ready to be implemented through Xilinx's Intellectual Property (IP) packages or pre-implemented bitstreams [64]. Next, although Xilinx's DPU exhibits more throughput than VTA, none of the architectures present information about performance predictability and power efficiency, which are our main concerns to deploy a successful multi-tenancy system. Furthermore, there are current efforts to improve VTA's throughput [77], so in that regard, neither selection constitutes a more interesting choice from our standpoint.

Justification We have to assess whether ISA-based architectures, well-regarded for their flexibility and competitive throughput, leverage the benefits from specialization in terms of energy consumption since their execution requires additional hardware that is not there to perform the computation but to orchestrate it. The list includes modules for loading and storing data, and

instruction fetching, decoding, and dispatching. We assume that these modules, present in any ISA-based accelerators, are the source of energy consumption overhead that calls such architectures into question. We argue that their common presence across designs allows us to extend the observations performed on VTA to other architectures that follow the same paradigm.

Next, and unlike Template-based designs, the performance predictability of ISA-based accelerators cannot be inferred from their architecture. Furthermore, as demonstrated in Clockwork [8], even an inherently predictable computing architecture can be wrapped in multiple layers of unpredictable entities, losing performance determinism. We believe that VTA's open architecture may give us an extra degree of freedom to enforce a deterministic behaviour, should there not be.

Finally, we aim to generate observations that are device-agnostic. That means that they can be extended to any type of compatible FPGA. Since VTA is a design that comes prepared to be applied in FPGA SoC devices, we assume that our conclusions can be extended to other FPGA platforms as long as:

- We can ensure that PS and PL can operate independently from each other.
- We decouple PS energy consumption from the power measurements.

3.2.2 Architecture description

Hardware architecture

A VTA program is composed of a set of instructions and micro-kernels. The set of instructions is very brief, including load and store data, micro-kernel execution call, and element-wise operations, such as *min*, *max*, *sum* and *bit rotation*. The micro-kernel simply defines a data access pattern for the matrix multiplication core, and it is flexible enough to implement different CNN primitives such as convolution, matrix multiplication or even transposed convolutions.

VTA's hardware architecture is parameterizable in terms of its bit width and concurrent computing capabilities. Figure 3.1 depicts our chosen accelerator's architecture that as an example features a data width of 8-bits. The entire accelerator is connected through an AXI interconnect router to a memory controller through a single coherent port [78]. The memory controller is in charge of managing the DRAM and memory coherency between the CPU and the PL, through the Cache Coherency Interface (CCI). The cache coherency interface controls memory regions that are marked as *shareable* between the PS and PL, making sure data at such regions are not masked by CPU's cache [79]. That means that when the PS writes input, instructions, or weights to the specific DRAM regions marked as shareable, the CCI triggers automatic cache flushes so

that all the information is written directly in the DRAM and is accessible by VTA. Furthermore, any modification in RAM performed by the accelerator through the coherent port triggers cache invalidation processes at the CCI. Hence, the CPU and accelerator are synchronized at all times. This approach is known as *hardware-enabled cache coherency*.

VTA's component modules and their functions are described in the following paragraphs.

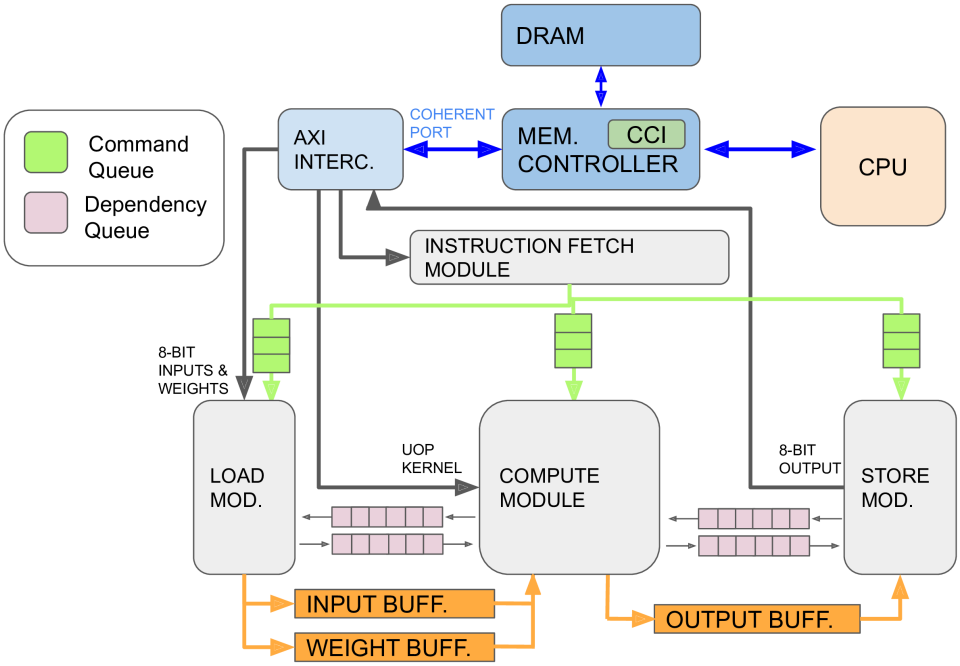

Figure 3.1: Hardware schematic of VTA. Adapted from VTA paper [70].

The **fetch** module extracts instructions from DRAM and dispatches them into queues for the different modules to work. Between modules, there exist dependency queues that enforce dependency between instructions, avoiding read-after-write or write-after-read data hazards. To execute a workload, the CPU deposits the sets of instructions and micro-kernels in DRAM and enables the fetch module to take the instructions from DRAM and command the other modules to perform the computation.

The **load** module retrieves data from DRAM according to the instructions it receives from its instruction queue. It will fetch input or weight data and store it in the corresponding queue for the compute module to take it.

Next, the **store** module retrieves data from the output buffer and writes it back to DRAM.

The **compute** module is in charge of executing micro-kernels and element-wise operations. It features a parameterizable General Matrix Multiply (GEMM) core and ALU core. The GEMM follows data access patterns specified by the kernel, while the ALU executes instructions dispatched from the fetch module. The size of the GEMM core defines also the data format and

access sequence of the accelerator. Since it can perform matrix multiplications in one clock cycle, its data units have to match the size of the compute core. Figure 3.2 shows a compute unit that can perform an 8-bit 16×16 matrix-vector multiplication in one clock cycle. Then, for this case, inputs are vectors of 16 units, and weights are tensors of 256 units (16×16). Intermediate results produced by the GEMM are stored in 32-bit, 16-element vectors at the register file. Next, the ALU unit can perform 16-element parallel, element-wise operations on the register file vectors, covering addition, activation, and bit-rotation operations. Finally, a memory port maps the 8 least significant bits of the register file vectors to the output buffer. This conditions the workload scheduling since no intermediate result from layer computation can be stored in DRAM. It also entails the need to perform bit rotation after a workload has finished execution to output the results in 8-bit format to DRAM. This challenge comes with the advantage of limiting data movement at the output port.

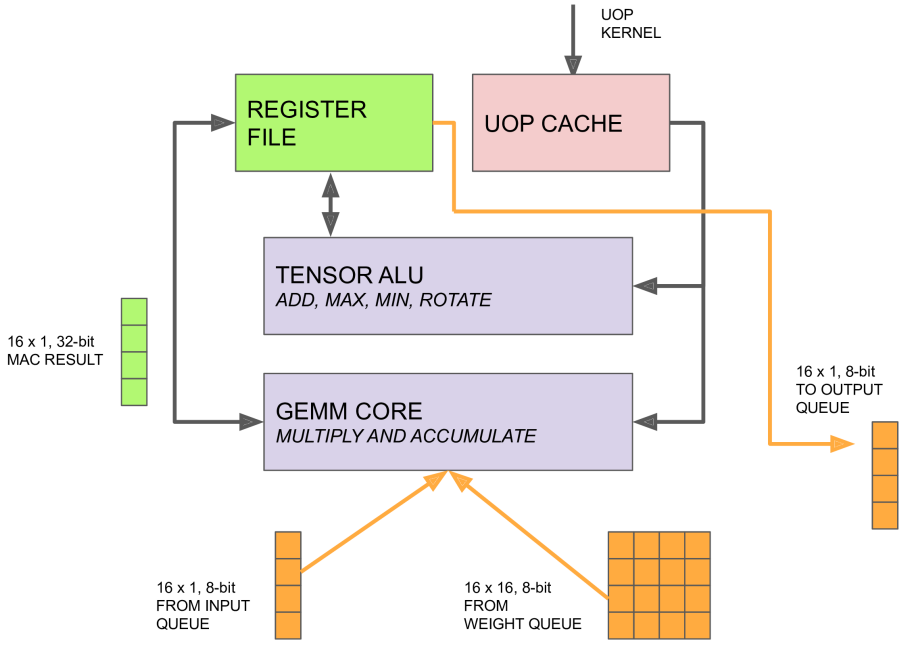

Figure 3.2: Compute module schematic. Adapted from VTA paper [70].

VTA software runtime

The software runtime controls and commands the accelerator. It is managed by the PS of the FPGA SoC. It receives a compiled VTA executable and then dispatches data and instructions in the DRAM locations that are expected by the accelerator. Moreover, it commands the fetch module to start once the information is arranged as expected. Finally, it retrieves the results that are written back by the accelerator once it has finished execution. Communication between

the accelerator and the PS is transparent, as it relies on a hardware cache coherency module to keep synchronization between DRAM and the PS cache.

DNN deployment

VTA leverages the graph optimizer/compiler TVM to generate executables and perform inference. There are several steps of optimization and adjustment needed for a graph algorithm such as a CNN to be turned into a deployable module.

Graph synthesis and quantization First, the TVM Relay front-end is used to process a CNN model trained in floating-point on a ML framework and construct a computation graph. Next, a quantization step is employed to turn weights and activations into 8-bit values. TVM offers post-training, data-aware quantization [80] to process the models and keep their original accuracy almost intact. Moreover, other graph optimizations can be done, like folding the batch normalization layer with the process of convolution, to avoid the computation of the former separately.

Tensorization and scheduling Then, an operator optimizer instance of TVM splits the computations needed to process each layer, into basic VTA operations, in a process that is known as *tensorization*. Considering the example illustrated before, tensorization converts convolutional and fully-connected layers into sets of 16×16 matrix multiplications that can be performed directly by the GEMM. Then, a threading agent splits computation from communication in the graph, so that both actions can be performed concurrently, hiding as much DRAM access latency as possible. Finally, a *schedule* of each of the layers is generated. The schedule describes how data is supposed to be accessed and processed by the core. Scheduling a workload is a crucial step because the register file cannot store all of the intermediate results and weights while performing the computation. The orchestration of the loading, storing, and computing data is therefore non-trivial. A separate process known as *autotuning* can try multiple scheduling strategies for each of the layers of interest in order to find the combination that offers the best throughput and yet generates valid results. In that regard, TVM offers a public database of optimal schedules of common workloads for each different device supported [81].

Executable generation Finally, the quantized, optimized, and tensorized graph is compiled into a deployable module, containing the executable and the weights, that can be handed to the VTA runtime. Figure 3.3 shows the deployment stack and the operations that are carried out on every layer.

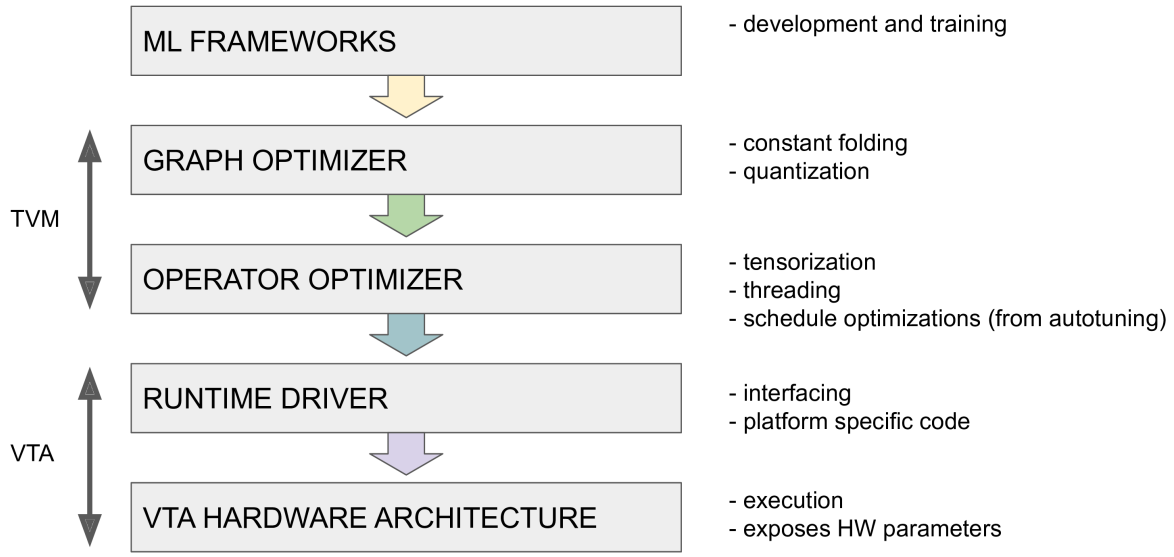

Figure 3.3: VTA Deployment stack. Adapted from [70].

Limitations It is worth mentioning that currently, the VTA toolchain only offloads to the accelerator those convolutional or fully-connected layers that feature a number of input and output channels that is multiple of the size of the GEMM, with other workloads being executed on the CPU. Our main concern is that the first convolutional layer of the majority of CNN models, that feature one or three input layers cannot be accelerated with the current toolchain. In the next section we show how we circumvented this and other limitations to benchmark VTA's performance under resource-sharing system use conditions.

3.3. Architecture modifications

Analyzing VTA's stack we found caveats that could be detrimental to a successful implementation of the FPGA accelerator in a resource-sharing system. We, therefore, leverage its open architecture to carry out three main modifications that improve performance under our use case of interest:

- We propose to extend the model support of the stack, to execute all the workloads of a model in the accelerator.
- We modify VTA's runtime to explicitly control the accelerator's memory management, to achieve predictable execution times.

- We rearrange memory transfer in the accelerator according to the tasks carried out by VTA's modules, to increase its effective DRAM bandwidth.

We present in more detail the use case and improvements over the next subsections.

3.3.1 Intent and use case

We want to assess VTA's accelerator feasibility in a resource-sharing system. We are interested on a CNN serving scenario where were a resource-sharing system is deployed at a datacenter. The centralized controller receives requests of multiple, diverse CNN or MLP models, and loads pre-compiled VTA software modules (weights and executables) to FPGA workers. It also dispatches the inputs to the workers to be processed. If the request rate of a specific model change, the controller can also load or unload new models in workers, should that be necessary. The FPGA worker can be implemented on a node with a PCI-express card or a SoC. The worker takes the executable and weights and executes the request using VTA's runtime.

VTA's accelerator, located in the FPGA card or SoC's PL, reads the executable and performs the tensorized matrix multiplications that implement convolutional layer or fully-connected workloads. It accesses DRAM to fetch inputs, instructions, weights and to store each layer's output that is either consumed by the next one, or constitutes the final output of the model. Finally, the PS or CPU recovers the results and sends them back to the centralized controller.

By analyzing VTA's stack, we found a few shortcomings in the design that may not pose a serious problem when used as a standalone inference device (its original purpose), but introduce limitations when considering its use as part of a multi-tenancy inference system. In the next subsections, we present the limitations we found in the VTA stack and the improvements deployed in the architecture in light of the use case of interest.

3.3.2 Extended model support

A resource-sharing system needs to support a variety of DNN models. In that regard, one of the concerning limitations of VTA's stack is that it is unable to compile full DNN architectures that feature layers with input or output channels that are not multiple of VTA's matrix-multiply core size, with such layers being executed in the PS.

We propose recycling an idea that is employed in Google's TPU design: zero-padding the input or output channels of convolutional or fully-connected layers so that they can be processed by the current stack. This strategy reduces the efficiency of the layer's execution, since the compute core is underutilized, but does not increase the number of operations that have to be

conducted by VTA as it features 16-channel parallelism. This approach extends the computing support of the accelerator to full models; the PS is not needed anymore.

Supporting a new assortment of layers brings the need of generating efficient VTA schedules for layers that were previously executed by the CPU. TVM's autotuning utility was modified to enforce *correctness* of the schedule. This was done due to the two-level task ISA nature of VTA, as its architecture can enforce dependency between instructions but not between micro-kernels. This means that some workload schedules can generate wrong dependencies, executing micro-kernels before the correct data is loaded, generating wrong results. During autotuning, the layer's parameters are loaded with random numbers extracted from a flat distribution. We modified the autotuning library to generate random values inside VTA's 8-bit range.

3.3.3 Deterministic runtime

As mentioned before, data synchronization between the PS and VTA is achieved, by design, through automatic cache flushing and invalidation processes that involve the SoC's CCI module; an approach called *Hardware-enabled Cache Coherency*. Next, memory in the accelerator is cached using a page granularity of 4 KiB, meaning that a data write operation from either the PS or PL will trigger multiple cache flush or invalidation process in an uncontrolled fashion. Furthermore, continuous DRAM use of the PS's Linux operative system means that the PL may need to wait sometimes for data to be duly copied from the cache, adding a degree of unpredictability to the system.

A resource-sharing system needs a predictable CNN inference stack to effectively schedule workloads, maximizing the system's utilization. We modified VTA's runtime to flush all shared regions automatically after data copy, and to invalidate data written by the accelerator after computation is finished. Such a strategy is called *Software-enabled Cache Coherency*. We expect this to increase the predictability of the system, as well as to slightly improve latency times since all necessary data will be always at DRAM when the accelerator needs it.

3.3.4 Task-based traffic orchestration

DNN workloads can be limited either by computation or communication speed. By looking at the original VTA design, we observed that the communication is carried out by a single memory bus. Since the memory controller of our SoC supports up to five coherent and non-coherent communication ports, we decided to employ different data lanes for different VTA functions. For each port, the memory controller handles request queues and in this case, serves requests from the different queues in a round-robin fashion [82].

We believe that this per-task data lane arrangement can help improve latency margins, because tasks are equally prioritized, avoiding many VTA functions colliding in one single data lane, increasing the effective DRAM bandwidth that is seen by the accelerator. In that regard, we assign a non-coherent port to input, weight, and micro-kernel loading functions, which comprise the majority of the data transfer, but since they exhibit dependency, they can share a single port. Using a non-coherent port is possible due to our previous software-enabled cache coherency scheme. Data I/O in non-coherent ports is faster since it does not generate any transaction on the CCI.

We then assign another non-coherent port to the instruction fetching task. This function is critical for the whole performance of the system, so limiting the access time overhead that the fetch module can experience should help to maximize throughput. Finally, a coherent port is used to store output data back to memory, triggering the synchronization process automatically between the PS and PL, allowing the PS to access that data as soon as possible. A schematic of the new VTA design is shown in Figure 3.4.

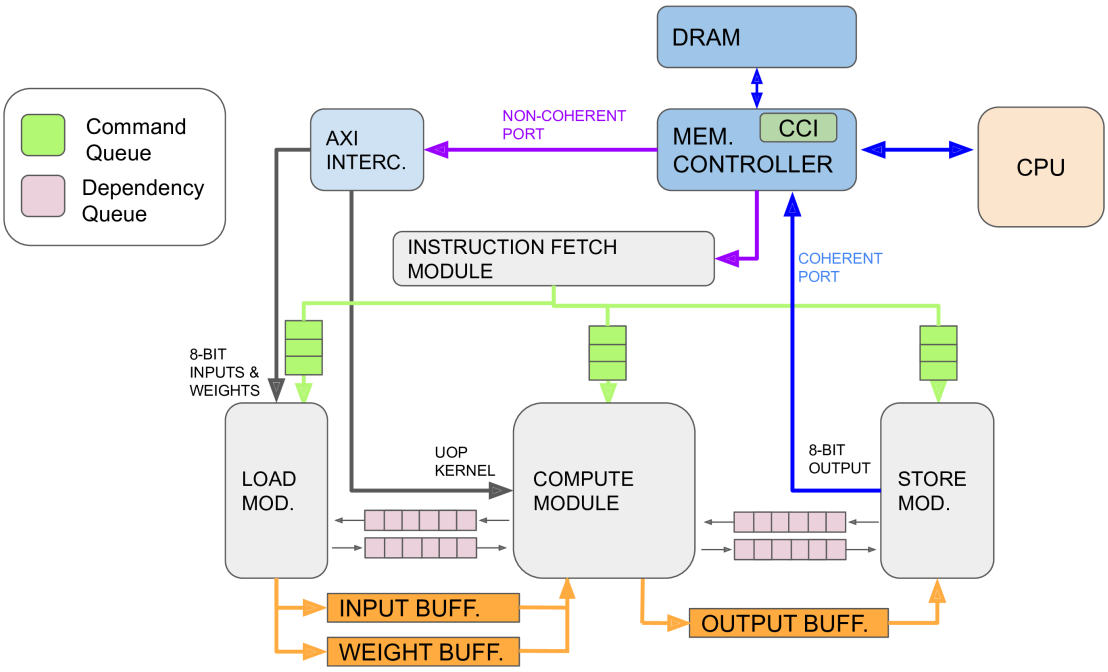

Figure 3.4: Modified High-bandwidth interface architecture for task-based memory access orchestration.

Chapter 4

Evaluation

We are interested in determining whether ISA-based accelerators offer performance predictability and energy efficiency, key decision factors that determine their feasibility for deployment on resource-sharing systems. Also, we want to ensure that such measurements can be extended to other FPGA platforms. We design a set of benchmarks that assess power consumption, throughput, inference latency distributions, and CPU DRAM write access latency distributions, the latter to guarantee that PS and PL are sufficiently decoupled from each other, and test them on VTA, our open ISA-based accelerator of reference. We assess the performance and predictability of the original and successively modified VTA designs. Then, we compare the best performing VTA design with a state-of-the-art GPU to assess their energy efficiency.

We empirically demonstrate that:

- ISA-based accelerators are not inherently deterministic, but predictable execution times can be achieved by informed design and memory management decisions.
- When compared with GPUs, deployment of ISA-based accelerators does not provide advantages regarding energy consumption.

4.1. Experimental setup

4.1.1 VTA Implementation

We employ a medium-sized FPGA SoC, a Xilinx Ultrascale+ ZU3EG, made with state-of-the-art 16nm FPGA fabrication technology [83], to perform the analysis of the performance and efficiency of VTA. The same fabrication technology is used for datacenter FPGAs. We use an Ultra96 FPGA SoC single-board computer [84], featuring a quad-core, 64-bit ARM Cortex-A53 Processor System (PS), running at 1.5 GHz, 2 GB of DDR RAM, and a VTA design loaded in the

Programmable Logic (PL) that features a GEMM to process 16×16 matrix-vector multiplications in 8-bit, running at 333 MHz, which gives a peak performance of 170.4 GOPS. Table 4.1 shows the resource footprint of our accelerator in our SoC. We execute a version of Linux for Xilinx devices called PYNQ [85], that provides memory management and the execution environment for the VTA software stack.

Device	Slice LUTs	Slice Registers	BRAM Tiles	DSPs
VTA 16 × 16 @ 333 MHz	32.37%	20.69%	63.19%	74.44%

Table 4.1: VTA Footprint in Ultra96 board.

4.1.2 Workload deployment

We connect the Ultra96 SoC board over ethernet to a host computer that compiles the workloads. Then, we deploy a VTA Remote Procedure Call (RPC) server on the Ultra96 that receives VTA's bitstreams and executables and launches execution, returning the result afterwards. VTA bitstreams are loaded in the PL using a library bundled with PYNQ. The executables are passed down to the VTA runtime for inference.

The executable generation and deployment is done over TCP/IP from a separate host computer running a RPC client, and using the TVM Python library to compile DNN models or generate single workloads, like convolutional or fully-connected layers that are useful for testbench generation. Figure 4.1 shows two different software stacks that can be used to deploy VTA workloads. The full TVM stack is used when compiling full DNN models. Otherwise, we employ TVM's operator collection library (TOPI) for standalone workloads.

4.1.3 Tooling

Workload set for our testbenches

We employ the convolutional and fully-connected workloads of a resnet18 classifier [86], extracted from Gluon CV Toolkit [87], denominated *resnet18_v1*, that accepts Imagenet-sized images [88] for the testbenches. We choose this model family because it is an industry standard, and includes many different types of convolutional layers (with kernel sizes of 7×7, 3×3 and 1×1, with and without stride) and a fully-connected layer at the end, presenting different computation and communication requirements; a thorough set of challenges for our accelerator.

We develop two different test scripts:

- A resnet18 inference testbench, employing the original TVM stack. Tasks such as the first

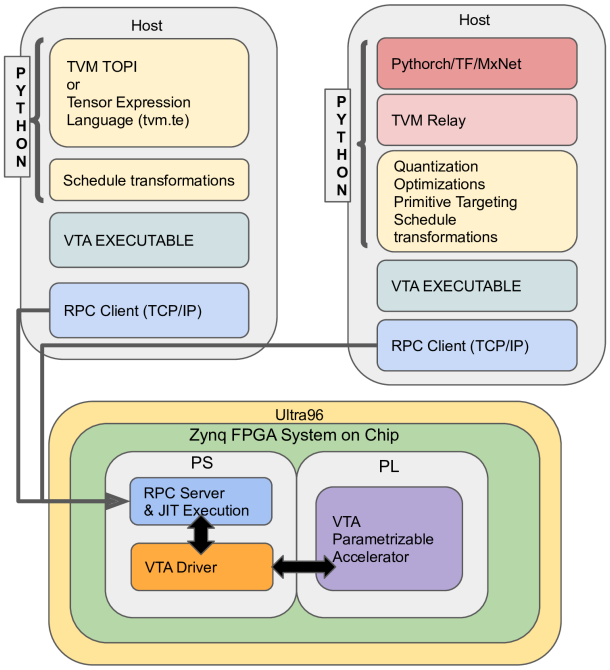

Figure 4.1: Two different software stacks are employed to generate VTA workloads.

convolutional layer, last fully-connected, data reshaping, and pooling layers are executed by the PS on the original stack.

- A testbench employing TVM's TOPI, that describes and sequentially executes all resnet18 workloads, giving us the possibility to decouple VTA's performance from that of the PS. The latency estimation of the model is composed by aggregating the latency of all the layers executed. We use this script to test all our architecture modifications, and characterize power consumption.

For each test case of interest, our scripts execute 10,000 resnet18 inferences and measure their latency, generating distributions that help assess performance and predictability for VTA's execution. We retrieve the latency marks of all the performed executions and store them on a CSV file.

Sysbench: DRAM Stress Utility

We are interested in replicating worker conditions under a resource-sharing system framework for our tests. In such a system, like Clockwork, the centralized controller exchanges inputs and models (in form of executables and weights) while the accelerator is performing inference. Realistic working conditions can help to determine the deployment challenges that come with ISA-based accelerators. Therefore, we modified a version of the *sysbench* benchmark tool [89],

to emulate them, by means of writing DRAM blocks at different data transfer rates specified by us.

Sysbench is a tool usually employed for database benchmarks. It contains a DRAM memory test intended to measure read and write latency, given a specified block size. Each test thread allocates a memory sector and performs I/O operation with it for a certain amount of time. Finally, a histogram for the access latency and the average data rate is displayed on the console.

The original memory test version employs threads that greedily try to saturate the RAM bandwidth by writing or reading a specific block size in DRAM. We modified the source code for sysbench to target a data transfer rate specified in the command line arguments. In this modified version, we employ the thread's time elapsed and cumulative data transferred records to calculate the amount of time it has to sleep before performing memory I/O again, to keep the desired data rate. Furthermore, it also generates a CSV file with the DRAM latency histogram, that can be transferred to the host computer thereafter. By employing this modified version of sysbench, we are able to fine-tune the amount of constraining on the DRAM that is used by both the CPU and the accelerator.

Power measurement by on-board telemetry

The SoC requires multiple voltage sources to power its different modules, namely the PS, PL, and external interfaces, such as USB, Ethernet, SD card, to name a few. The board's programmable power regulators generate all on-board voltages from an external 12V supply, control current levels, and provide access to power telemetry through a PMBus interface [90] that is routed to the PS. PYNQ provides a Python library to access and monitor the power regulator status. Therefore, it is possible to access each power converter voltage, current, and power measures to supervise the Ultra-96 energy consumption.

Automatization over SSH

We employ SSH automatization to launch VTA workloads, and SoC power and DRAM stress tests from a single script that is executed in the host computer. The script also collects the results generated by all the sub-processes.

4.2. Latency benchmarks

In this work, we performed three major modifications on the VTA design, that aim to extend its model support, increase its predictability, and improve its memory access.

In this section, we test VTA's performance and present inference latency distributions under increasing DRAM memory bandwidth constraints. We programmed our modified *sysbench* version to write 16MiB blocks to DRAM, occupying seven different bandwidth levels, starting from 100MiB/s, up to 2GiB/s, and then as much as possible, yielding a maximum average data write rate of 2.7GiB/s when performing inference, and 3GiB/s when VTA is not in use.

The modifications presented in Section 3.3, were deployed and tested on an aggregated approach, one on top of the other, since they serve the purpose of increasing VTA's overall performance. Therefore, they are presented progressively throughout the next subsections.

4.2.1 VTA-only execution: Extended model support

Implementation

The first experiment aims to produce a performance baseline of the current accelerator architecture, as offered in its repository, and also a new benchmark that can characterize the accelerator's behaviour as a standalone device, with no workload being executed on the CPU, and with the full support of all convolutional or fully connected layer by means of zero-padding the input or output channels, if needed.

Considering the resnet18 CNN model, there exist two layers that are not compatible with VTA: the first convolutional and last fully-connected layer. We therefore circumvent this limitation by zero-padding the number of channels to a multiple of 16.

Table 4.2 presents the convolutional and fully-connected workloads of resnet18, our CNN reference model. Highlighted in bold we show the layer sizes that were padded to fit that particular workload in VTA. The repeat factor refers to how many times that particular workload size is executed in the resnet18 model.

Supporting new padded workloads on VTA requires the user to provide a suitable schedule. The schedules for our resnet18 padded layers are not included in TVM's tophub, so an autotuning process is required. We use the autotuning script to tune specifically the first convolutional and the last fully-connected layer.

We compile resnet18 on the original TVM stack (PS executes first and last layers), and compare the results with our TOPI workload set that includes the padded first and last layers.

Results

We execute the inferences on the original stack and our workload set for each DRAM stress level case and generate latency distributions. We show that:

Workload	Feat. Map Size	IN CH	OUT CH	Kernel	Padding	Stride	Repeat
CNV1	224×224	3 **(16)**	64	7×7	3	2	1
CNV2	56×56	64	64	3×3	1	1	4
CNV3	56×56	64	128	3×3	1	2	1
CNV4	56×56	64	128	1×1	0	2	1
CNV5	28×28	128	128	3×3	1	1	3
CNV6	28×28	128	256	3×3	1	2	1
CNV7	28×28	128	256	1×1	0	2	1
CNV8	14×14	256	256	3×3	1	1	3
CNV9	14×14	256	512	3×3	1	2	1
CNV10	14×14	256	512	1×1	0	2	1
CNV11	7×7	512	512	3×3	1	1	3
FC1	1×1	512	1000 **(1008)**	1×1	0	1	1

Table 4.2: Gluon's resnet18 workload description for Imagenet dataset.

- The average latency is lower with our proposed approach that executes all the model's workloads in hardware.
- Latency variation is significantly lower when not using the PS.

Figure 4.2 compares resnet18 inference latency distributions of VTA's original stack and our set of workloads prepared for VTA-only execution.

It is possible to see that the mean average and range (difference between maximum and minimum value) of the inference distributions increases in both designs when the DRAM bandwidth is progressively constrained. We understand that sysbench interferes with VTA's DRAM access, causing some workloads to suffer data starvation, increasing waiting times for data. Furthermore, we can observe a wider deviation in the original VTA stack, which we attribute to the workloads that are executed in the PS. In a resource-sharing system, a thread that loads models (simulated in our scheme by sysbench) would compete with those that perform inference for CPU time. Figures 4.3 and 4.4 compare the latency distributions of the layers originally executed in the PS against VTA (denoted CNV1 and FC1 in Table 4.2), illustrating this phenomenon; the PS may be occupied loading models when a workload arrives, or the operative system schedule may switch between computing and data transfer threads in the middle of an execution, increasing waiting times and widening the latency distribution, even at the very slow rate of 100 MiB/s.

With our VTA-only approach, the inference hardware does not get interrupted by any other task, thus latency deviation can only occur due to data starvation. Then, the impact on the

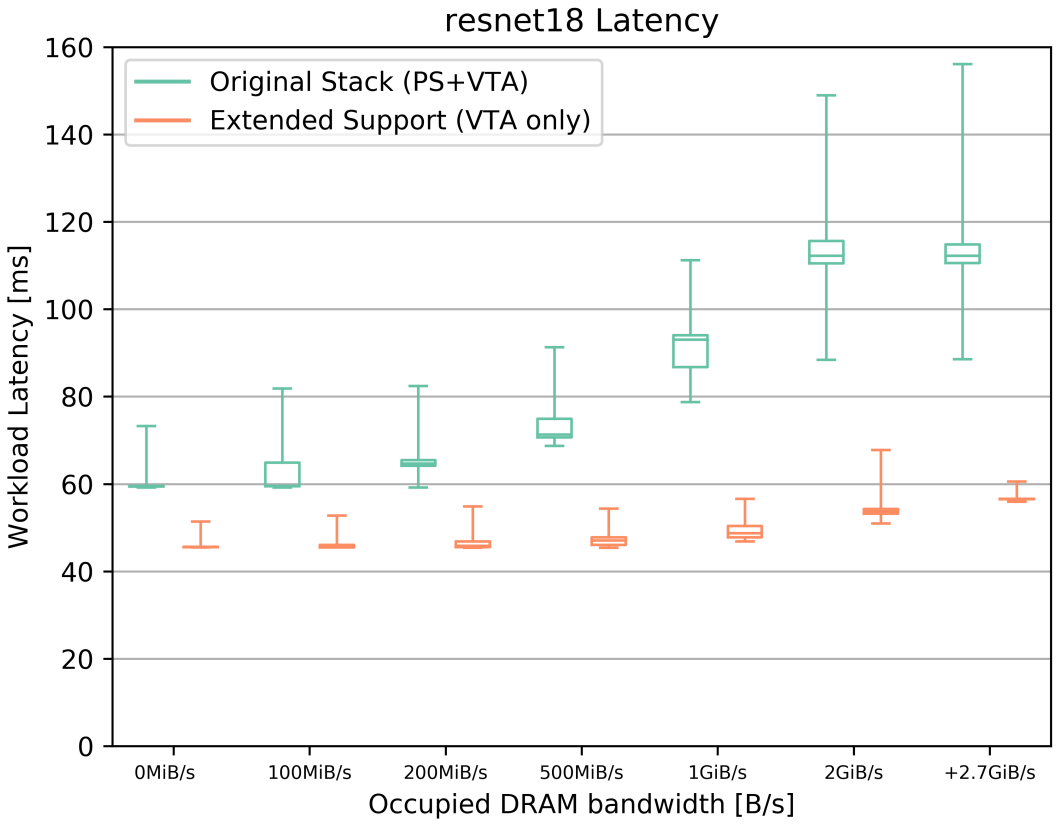

Figure 4.2: Latency distribution comparison between the original and the extended support approach.

latency variance is smaller than in the previous case.

Next, it is interesting to compare the average performance of VTA and PS in both CNV1 and FC1 workloads. For CNV1, the mean inference time is smaller for VTA, even with the accelerator computing the 16 padded input layers and the PS only 3. However, the same does not hold for FC1, with the PS showing a significantly smaller average inference time. This showcases VTA's ability to leverage data reuse typical of convolutional layers, like CNV1. The FC1 workload is strongly memory-bounded when being executed on VTA, since fully-connected layers exhibit no weight reuse. The accelerator nevertheless, presents a much smaller latency deviation under bandwidth constriction than the PS counterpart, as computation does compete with data loading when executing the layer with VTA.

By deploying all workloads of a CNN model with VTA, we significantly decreased latency variability and improved average inference time on the platform with respect to the original stack. Nevertheless, the minimum latency range of about 6ms we observe in the latency distributions with this improvement is still not acceptable for a deterministic inference platform required in a

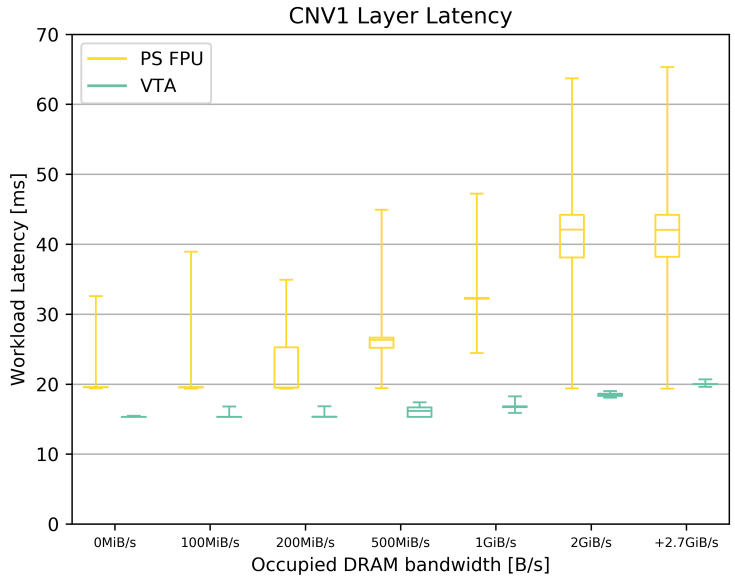

Figure 4.3: Detail of CNV1 execution on both PS and the accelerator.

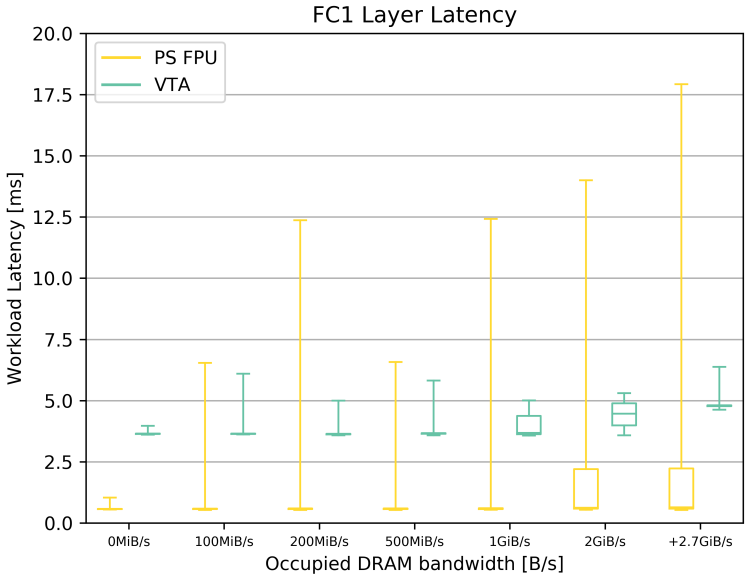

Figure 4.4: Detail of FC1 execution on both PS and the accelerator.

resource-sharing system.

4.2.2 Predictable Runtime

We modified VTA's runtime to perform memory update operations between the shared DRAM and the PS's cache, without relying in automatic hardware strategies to do so. We intend to reduce latency variability by disabling hardware cache coherence.

In this subsection, we present a set of resnet18 inference distributions conducted with our

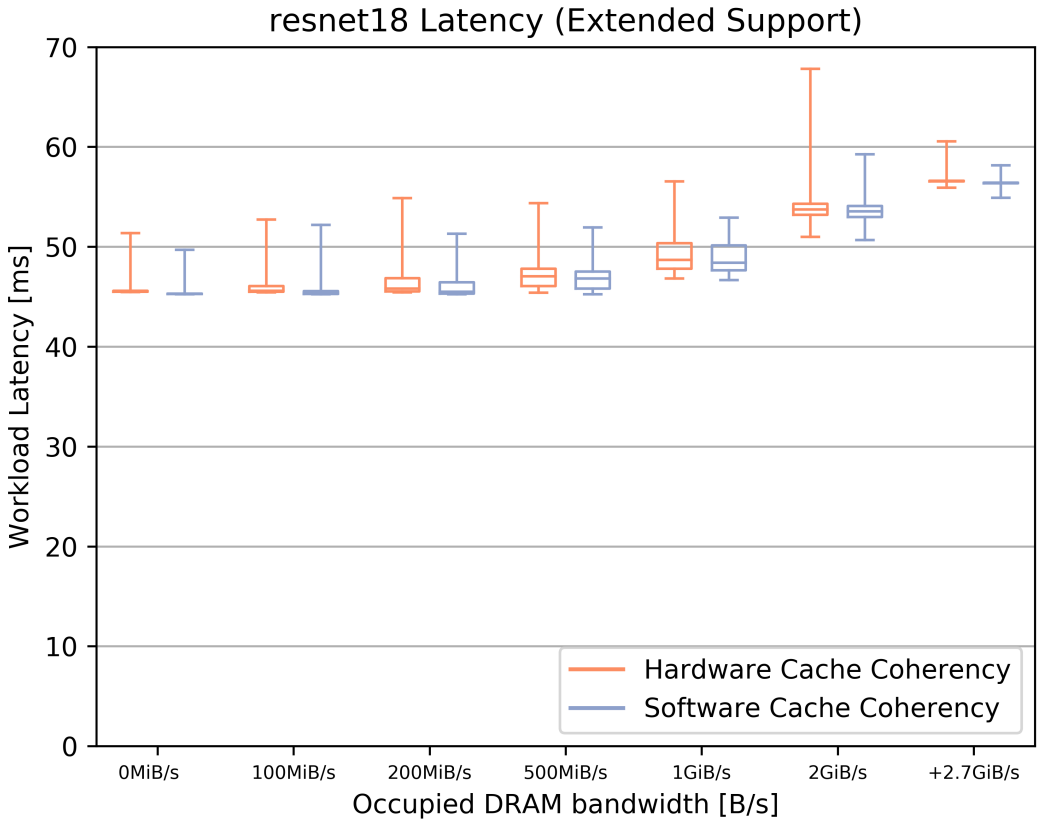

Figure 4.5: Comparison between latency distributions obtained using Hardware and Software cache coherency strategies.

extended model support approach, where we compare performance and latency deviations between the original Hardware Cache Coherency runtime (denoted HWC) and the explicit Software Cache Coherency one (SWC). We generate latency distributions for both cases, under different DRAM bandwidth occupancy conditions. As shown in Figure 4.5, we achieve better inference predictability than with the original approach under all DRAM bandwidth stress levels.

The DRAM holds VTA's inputs, weights, and instructions needed to perform inference. When the PS stores such data, some information may end up being cached and not actually written to DRAM. We believe that by explicitly controlling the memory transfer between the PS and PL, we manage to update all cached pages in one single operation, minimizing the transfer overhead that would be generated should we update each of the pages only when they are needed by VTA, as it happens in the Hardware Cache Coherency approach.

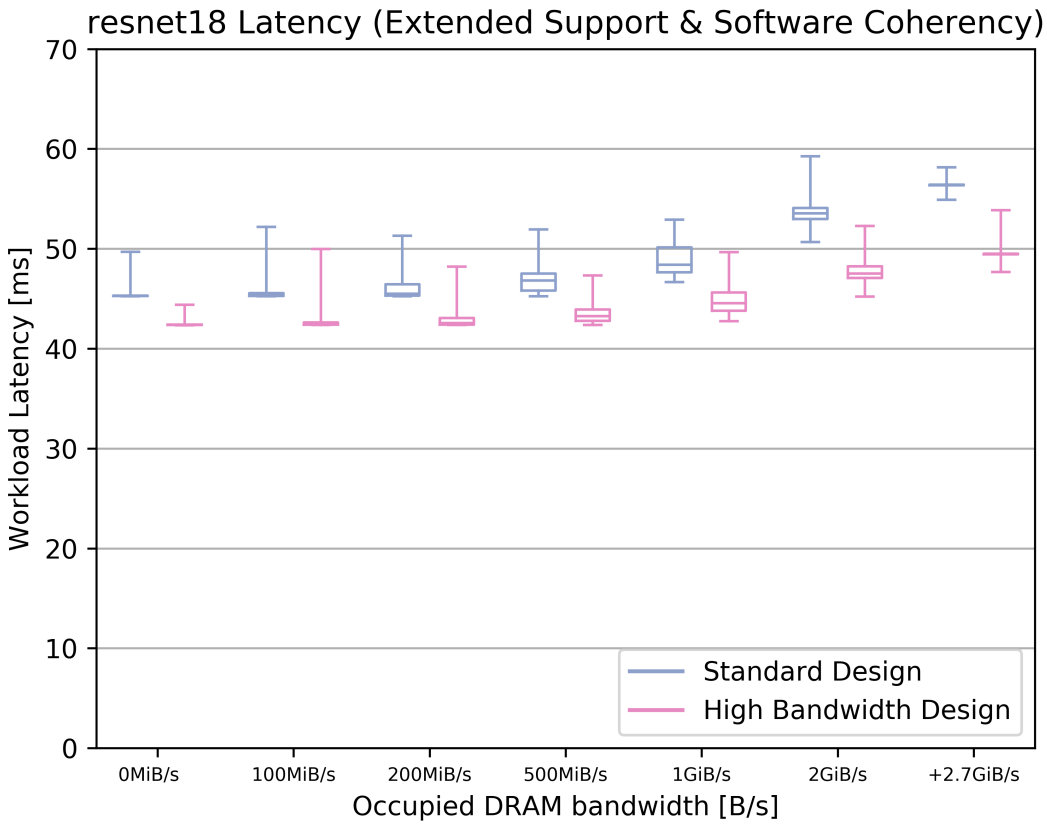

Figure 4.6: Comparison between latency distributions of the standard and high-bandwidth accelerator designs.

4.2.3 High Bandwidth design

By increasing the effective DRAM bandwidth of our accelerator, we expect that the memory-constrained workloads of resnet18 show higher performance than with the previous designs. It is worth noting that this new design has an FPGA resource footprint that is similar to the original one, shown in Table 4.1, since we only modify the data lane connections; our design employs the same computing and local memory resources of the original design.

We deploy the modified Software Cache Coherency runtime with our high bandwidth accelerator bitstream and obtain performance and latency distributions. We compare these with the ones of the original VTA accelerator architecture, under the same different bandwidth constraint cases. Figure 4.6 shows the latency distributions obtained by both the original and the High Bandwidth design that employs task-based memory orchestration.

It is possible to see that with our design, the average latency surpasses that of the original design for all DRAM bandwidth stress cases. Furthermore, when the system is not stressed, our design exhibits a total latency variability of 2ms.

Finally, it is worth noting that under heavy DRAM use cases, the high-bandwidth design produces a wider latency distribution than its counterpart. We suspect this is due to the DRAM access orchestration that is carried on at the DRAM controller of the SoC, which now handles many data ports connected to the accelerator and the PS. A performance tuning strategy assigning Quality of Service (QoS) priorities to the data ports may help mitigate this result, but it is out of the scope of this work [79].

4.2.4 Memory interface

We also want to assess the level of isolation between the PS and PL with VTA's accelerator loaded, to determine if PS and PL could work independently.

Figure 4.7 shows the write access distributions generated of the PS writing to 16 MiB blocks, evaluating the performance of a worker loading inputs or models into a DRAM that is accessible by the FPGA. In black, we show a baseline distribution, obtained when writing blocks with unlimited bandwidth occupancy, with a final average data transfer rate of 3 GiB/s. The other distributions were obtained while running inference, from each of the seven stress levels tested, and each of the three VTA-only experiments performed.

We see that the average write block time slightly increases when executing VTA, since DRAM is also accessed by the accelerator. Nevertheless, the same average waiting time is observed for each of the stress levels and VTA stack version tested, with maximum waiting times varying without a perceivable pattern from design to design and experiment to experiment.

When considering a resource-sharing system, from these results we can conclude that the CPU performance when writing models and inputs to the accelerator's DRAM does not suffer greatly while the accelerator is in execution. Furthermore, DRAM access is inherently unpredictable due to other processes present in the operative system that is run by the CPU; a finer control of the threads by the operative system can help to put an upper bound to DRAM access unpredictability.

4.2.5 Overview and discussion

Determinism was never a goal for the original design, but by leveraging VTA's open architecture we performed informed modifications in the stack that boosted VTA's predictability, latency, and throughput, and since we improved upon a single computing core design, latency and throughput do not trade-off one for the other.

Next, we see predictability as an effect of improving memory management and concentrating all the computation on the accelerator, without relying on the PS to compute any layers.

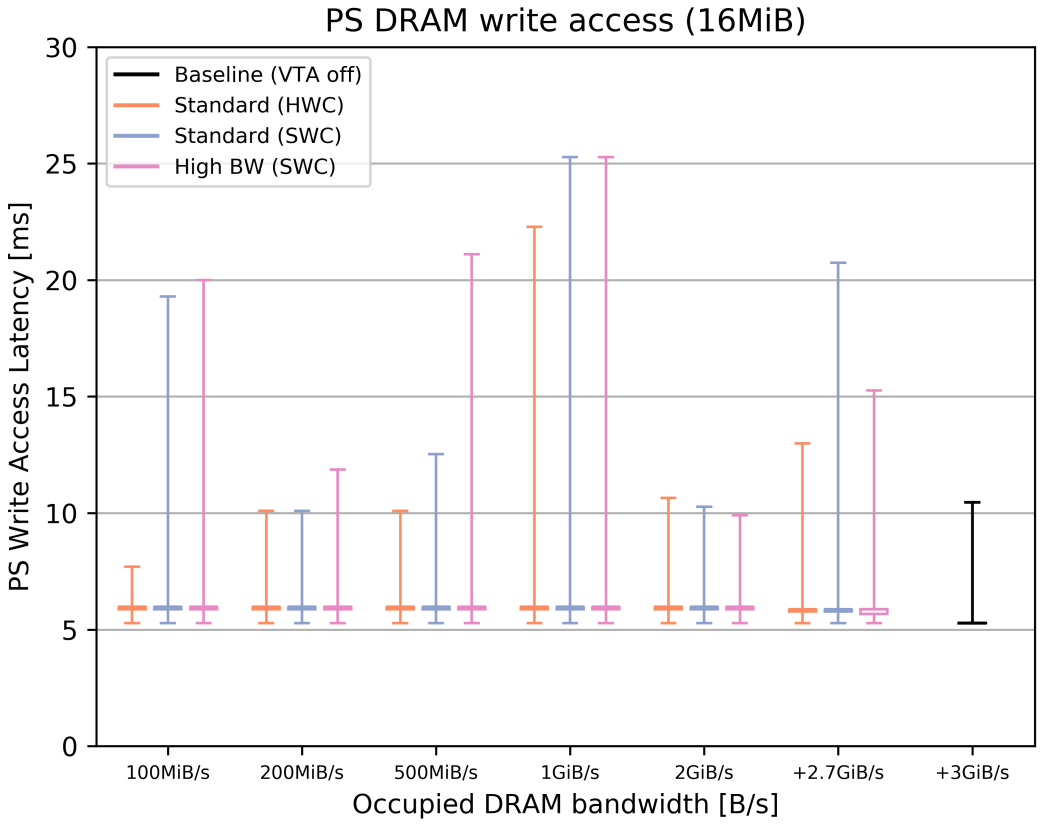

Figure 4.7: DRAM write access latency distributions seen from PS side when VTA executes inference.

Table 4.3 presents the range of the inference distributions obtained for each of the designs and experiments performed.

Constraint	Original Stack	Extended (HWC)	Extended (SWC)	High BW (SWC)
0MiB/s	14.11	5.88	4.43	**2.05**
100MiB/s	22.63	7.29	6.92	7.63
200MiB/s	23.22	9.43	6.04	5.87
500MiB/s	22.54	8.97	6.67	4.98
1GiB/s	32.41	9.73	6.24	6.93
2GiB/s	60.53	16.83	8.58	7.09
+2.7GiB/s	67.500	4.630	3.23	6.19

Table 4.3: Range of resnet18 latency distributions, measured in milliseconds.

As expected, the most predictable set of inferences is obtained for the High Bandwidth design, using Software Cache Coherency, and when there is no DRAM access from the PS while

VTA is performing inference. Since all ISA-based accelerators rely heavily on DRAM to fetch instructions and data, this outcome allows us to believe that when employing them on a resource-sharing system, inferences should not be conducted while there is data being written to the FPGA DRAM. Therefore, new inputs or models should be written after the accelerator has finished its execution, which in some cases might reduce resource use considerably.

Regarding performance, Table 4.4 shows the average throughput measured in Giga Operations per Second (GOPS) of our workload set when being executed on the different VTA designs that were tested. The *SpeedUp* column shows the speed gain between the original and our fastest design. In bold, we show the most significative speed-ups of the workloads, that at least achieve an extra 10% of performance with respect to the original design. The table includes the effective GOPS achieved for each layer, without considering the padded channels in CNV1 and FC1 that add nothing to the actual resnet18 performance. For these layers, we show the overall performance of the calculation (including the padded layers) in italic.

	Design			SpeedUp (%)
Workload	Standard (HWC)	Standard (SWC)	High BW (SWC)	
CNV1	15.40 *(82.18)*	15.42 *(82.28)*	17.97 *(95.89)*	**16.68**
CNV2	124.94	125.22	125.70	0.61
CNV3	103.84	119.27	120.45	**16.00**
CNV4	18.89	22.59	24.49	**29.65**
CNV5	138.81	140.05	140.62	1.30
CNV6	109.14	134.87	136.28	**24.87**
CNV7	21.08	26.56	29.64	**40.61**
CNV8	148.08	149.8	150.36	1.54
CNV9	77.25	115.97	121.7	**57.54**
CNV10	21.8	24.23	26.51	**21.61**
CNV11	113.74	137.73	144.14	**26.73**
FC1	0.28 *(0.28)*	0.29 *(0.28)*	0.31 *(0.31)*	**10.71**

Table 4.4: Workload performance comparison between VTA designs. Measured in GOPS.

Finally, considering the layers' characteristics showcased in Table 4.2, it is possible to see that the three layers that were not significantly accelerated were those that exhibit the biggest data reuse patterns, featuring a stride of 1, padding of 1, and 3×3 kernels. We can infer that all other layers, that feature higher stride or have no data reuse, as in the case of 1×1 kernels, were memory bounded by VTA's architecture, and achieve better performance when data access

becomes more deterministic and the effective DRAM bandwidth increases.

By analyzing the architecture improvements and their effect on the extracted inference distributions, we can conclude that for ISA-based architectures, predictability does not come for free, requiring careful memory management, design decisions, and data loading and inference orchestration to achieve it. The same holds for the process of loading data and models into the accelerator. Nevertheless, we believe that by making informed decisions, it is feasible to deploy ISA-based architectures in a resource-sharing scheme, from a predictability standpoint.

4.3. Power efficiency Comparison

We want to assess power consumption of VTA, and compare it with a state-of-the-art GPU executing the same workloads.

4.3.1 Implementation

For comparison, we select a state-of-the-art Nvidia Tesla V100 datacenter GPU, made with 12 nm fabrication technology [50], running inference in 8-bit, using a batch size of 1. We decide to measure performance at this batch size since it supposes a best-latency scenario, which resource-sharing systems often employ when there are no similar requests in queue for the model that is about to be executed. Although we know that our small version of VTA cannot be compared in terms of throughput with the GPU, we believe that a comparison can be established in terms of efficiency, as our design will process information at a slower pace but will also consume less energy to do so.

To correctly characterize the energy consumption of our design, we exclude the power measurement of the PS, measuring only the PL and DRAM, since the accelerator's data and instructions are stored there, and frequent accesses are performed throughout inference. For this test, no stress is applied to the DRAM port; we assume the DRAM energy consumption generated by the PS to be negligible when idle, waiting for inference to finish. We employ the high-memory bandwidth version of the accelerator as it presents the best performance of all tested iterations.

A schematic of the power distribution of the Ultra96 board is shown in Figure 4.8, adapted from the complete schematics that can be found on the manufacturer website [91]. The schematic depicts the Power Management Integrated Circuits (PMIC) that generate the voltage sources needed by different parts of the SoC in the board. PS and PL are powered from different PMICs, that receive a 5V source generated from another integrated power converter. We measured power delivered from the **INT**, **AUX**, and **PSDDR** voltage sources (highlighted in blue), accounting for consumption at the PL, DRAM interface, and DRAM chipset. We evaluate this while exe-

cuting our resnet18 workload set ten thousand times. Efficiency is obtained then by dividing the number of operations performed with the energy consumed during inference.

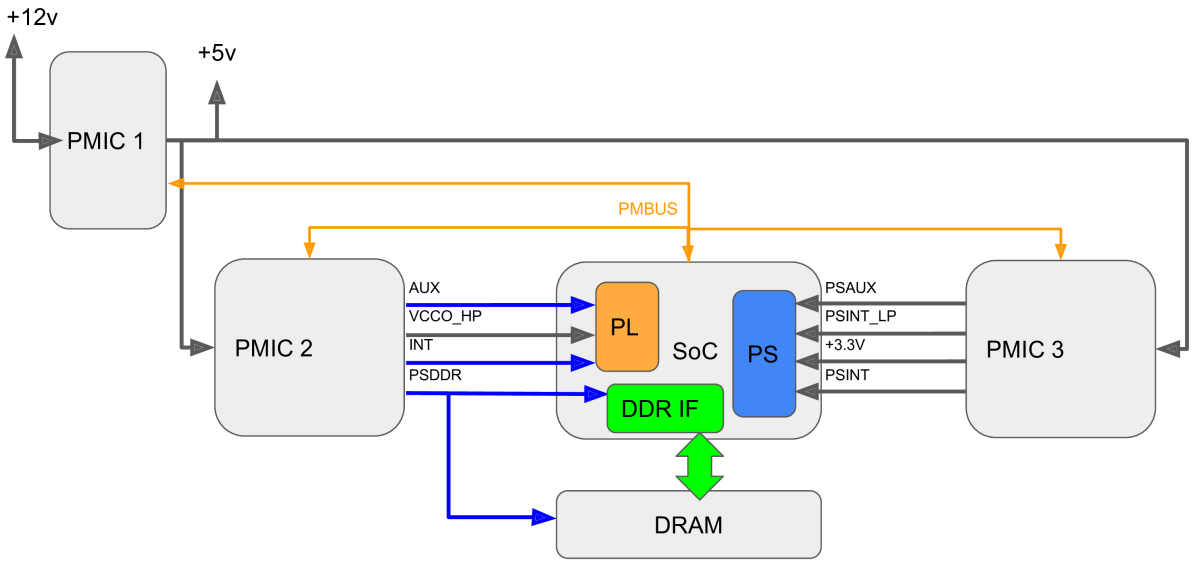

Figure 4.8: Power distribution architecture in Ultra96 board.

Finally, to characterize the GPU power efficiency, we use TVM to perform autotuning of a resnet18 model, for inference in 8-bit, using a batch size of 1. After the optimal schedules were found, we execute resnet18 inferences while measuring power from Nvidia's system management interface *nvidia-smi* [92]. We can estimate efficiency by observing the average performance of the complete run provided by TVM, and average the power measurement shown by the utility over the inference time.

4.3.2 Results

For VTA, we generate a power consumption logger on the PMBus, and store the power output measure that is generated by our Ultra96 SoC FPGA board, while running 10,000 iterations of each of the resnet18 workloads using the high bandwidth design with software-enabled cache coherency. Figure 4.9 shows the addition of the power consumption measured at the power sources INT, AUX, and PSDDR, accounting for energy consumption at the PL and DRAM, when executing our workload set. Since our testbench executes sequentially each of the resnet18 layers 10,000 times in a loop, we can see the evolution of the power consumption for each of the layers. The small drops are attributed to VTA being idle when waiting for the next workload to be loaded. We attribute the spikes to the low power resolution that is measured at the PMBus.

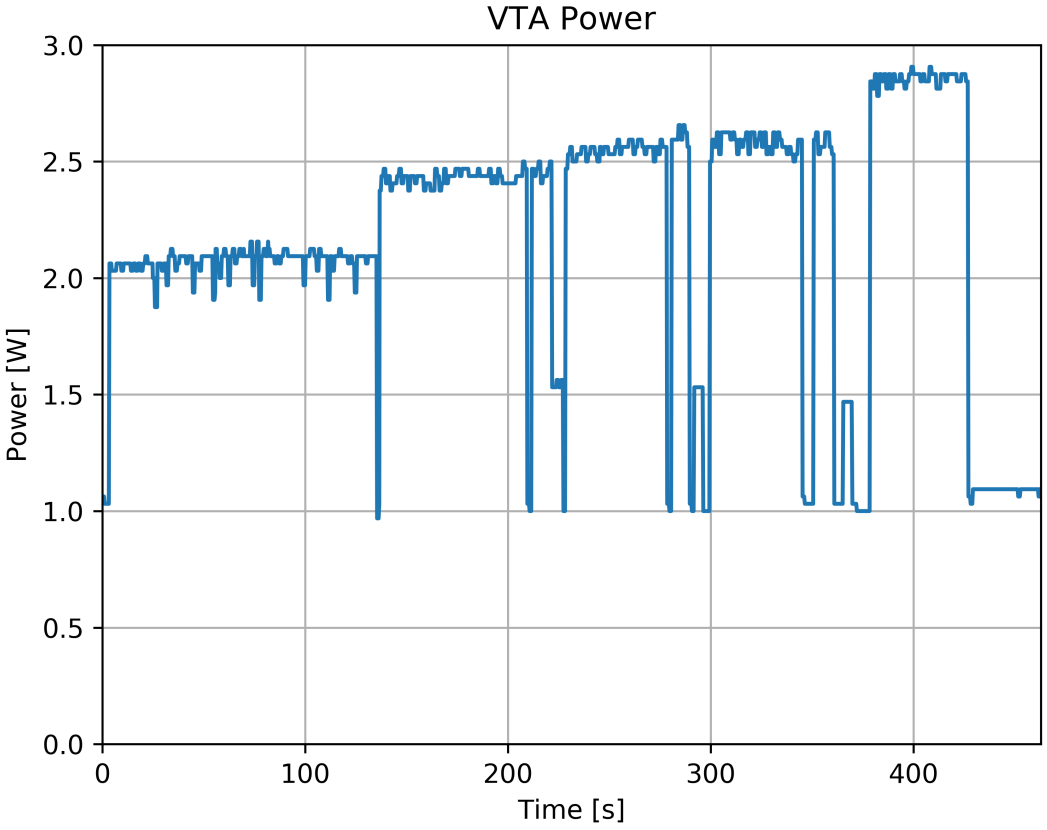

Figure 4.9: Total power consumption of PL and DRAM Interface evaluated when executing our workloads.

By knowing the number of operations carried out by the accelerator, we can then integrate the power consumption over time to obtain the amount of energy consumed in Joules (J), to finally determine the efficiency η_{VTA} of our accelerator when running resnet18, measured in operations over energy, OP/J or operations per second (performance) over Watt OPS/W, which are equivalent, as shown in Equations 4.1 and 4.2, where $p(n)$ is the power read in Watts, $t(n)$ is the time step assigned to each power value, N is the number of power measurements taken and K is the total number of operations carried out by resnet18.

$$Energy[J] = \sum_{n=1}^{N} p(n) \cdot t(n) \tag{4.1}$$

$$\eta_{VTA} = \frac{10000 \cdot K}{Energy} = 36.15\ GOPS/W \tag{4.2}$$

We proceed differently for power measurement at the GPU, employing a TVM inference script that compiles and then executes 10,000 times the same resnet18 model using 8-bit inference, and batch size of 1, using the optimized schedules obtained with TVM's autotuning process. After

execution, the script prints the overall performance obtained by the model in GOPS. While inference is taking place, we query the utility *nvidia-smi*, obtaining the GPU power measurements, shown in Figure 4.10. Since this technique does not guarantee a fixed sampling rate, nor we do generate time steps for each measurement, we decide to estimate a worst-case scenario by taking the highest power reading obtained during execution, and use it to divide the model's performance to obtain the model's efficiency η_{GPU} when executing in the Tesla V100, as shown in Equation 4.3

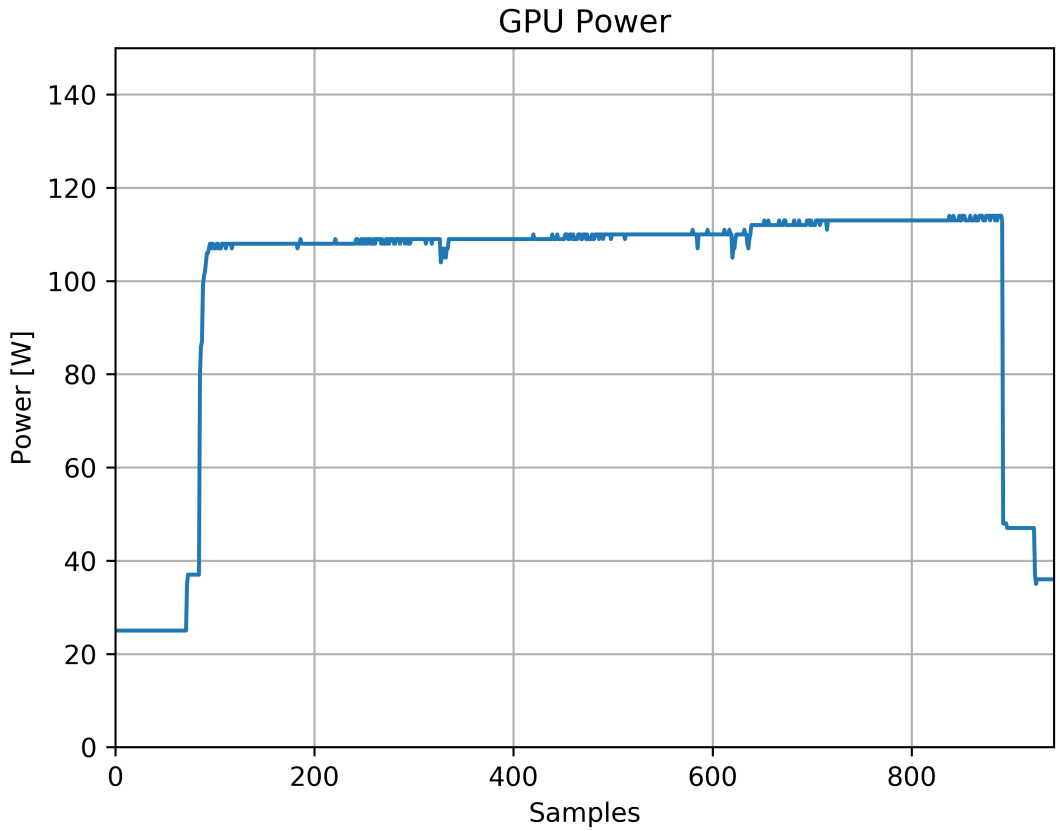

Figure 4.10: Log of the Nvidia V100 GPU power consumption while executing 10,000 inferences of resnet18 (8-bit, batch size of 1)

$$\eta_{GPU} = \frac{Performance[GOPS]}{Power[W]} = \frac{5838}{114} = 51.21\ GOPS/W \tag{4.3}$$

4.3.3 Overview and discussion

Table 4.5 shows the results of our resnet18 performance and efficiency estimation for both devices, showing that the GPU is roughly 40% more efficient than our ISA-based FPGA accelerator for this particular task, even under a worst-case scenario power measurement, and with the less

efficient inference scheme of a batch size of 1.

Device	Performance [GOPS]	Efficiency [GOPS/W]
VTA, High BW (SWC)	109.5	36.15
Tesla V100, batch = 1	5838	51.21

Table 4.5: Performance and efficiency of VTA and GPU when executing resnet18.

The nature of VTA's general matrix-multiplication compute core does not allow the design to leverage the benefits from specialization, but as mentioned before, there exist several digital design techniques that can help increase its efficiency, and that are not implemented on VTA's current version, such as efficient mapping of the computing resources [65], or generating a specialized compute core to efficiently process the input layers of CNNs that feature a low input channel count such as CNV1 in our case [64]. For the case of VTA, we also believe that further efficiency could be obtained from better memory bandwidth and buffer management, which might allow memory-constrained layers, such as CNV4, CNV7, CNV10 and FC1 in our testbench, increasing their throughput.

Next, regarding performance, it is evident that the small 16×16 GEMM core of VTA is not comparable with a state-of-the-art GPU. Nevertheless, since the compute core is parameterizable, peak throughput can be increased by placing a bigger GEMM on a bigger FPGA device, or by multi-threading, deploying multiple compute cores on a single device [64], [77]. This approach combined with a faster DRAM memory, such as one with a HBM interface [93], would allow for a higher effective throughput. Nevertheless, since predictability and efficiency are our main concerns, increasing performance performance of VTA by reshaping its GEMM core and employing a different memory interface on another FPGA card is outside the scope of this work.

As mentioned before, to our knowledge this is the first efficiency estimation of an ISA-based architecture. The only other related measurement we found is the power rating of Xilinx's commercial ISA-based DPU [64], which is 225W, comparable to the 250W of the Nvidia Tesla V100. From our efficiency benchmarks, the possible improvements discussed, and also considering the similar power ratings between commercial ISA-based accelerators and GPUs, we estimate that the former, using current FPGA technology, could be in a position of achieving a power efficiency comparable to a state-of-the-art GPU running with a batch size of 1 for DNN models that exhibit similar computation and communication characteristics as resnet18. When assessing typical GPU usage in a resource-sharing system, the centralized controller will always try to batch requests in order to maximize device use and increase efficiency; with batch sizes of 2 or 3 being common for typical request sets [8]. This means that current ISA-based accelerators may not be

desirable when compared with state-of-the-art GPUs due to poorer energy performance.

Finally, although both fabrication technologies of the GPU (12 nm) and the FPGA (16 nm) are state-of-the-art at the datacenter in their respective fields, they are present a different power performance. We compare them because that is what the industry is currently employing [14], [94]. We would expect FPGAs with newer fabrication technology to show better energy performance, but that is outside the scope of this work.

4.4. Conclusions

We have empirically shown that, for the current generation of FPGAs, ISA-based accelerators are not suitable for the multi-tenancy DNN serving paradigm of interest. Although we have proved that they could be deployed in a resource-sharing scheme, being quick to switch tasks, and presenting predictable inference when communication between accelerator and DRAM is well managed, they cannot stand against current GPUs in terms of power efficiency.

Finally, considering current designs, we understand that FPGA accelerators could prove themselves better than GPUs at the datacenter only for specific serving cases where there is a narrow selection of memory-bounded workloads, or when there is no need to switch models at all; cases where dominant workloads are present.

Chapter 5

Summary

This chapter presents a summary of the problems solved compared to the objectives presented in the introduction and in the Thesis Proposal Form. We then point out opportunities to move forward, motivating future work.

5.1. Problem statement

System designers are opting for multi-tenancy schemes on many cloud serving tasks because they optimize resource use, increasing the efficiency of the deployed application. The same holds for DNN serving, where recent strategies rely on predictable GPU workers to schedule workloads ahead of time to perform inference.

On the other hand, FPGAs are excellent devices to accelerate algorithms and are typically used for highly efficient, real-time applications, including DNN inference. Furthermore, cloud providers are featuring FPGAs on their options, since they can be reconfigured for potentially very different tasks.

We were interested in assessing whether a multi-tenancy system could be compatible with FPGA DNN acceleration at the cloud, to leverage the best of the two worlds.

5.2. Results

Our initial premise was that FPGAs would offer less energy consumption than canonical computing architectures also when implemented on resource-sharing systems that serve arbitrarily diverse CNN and MLP models. Between different FPGA accelerator designs, we were interested in ISA-based accelerators because they are quick to switch models, which is mandatory for multi-tenancy DNN serving. We employed an ISA-based CNN accelerator as a reference design, and found that such accelerators can have predictable latency, and could therefore be implemented in

a resource-sharing serving scheme, but they do not offer the efficiency we expected. The caveat rises from the fact that multi-tenancy requires a high amount of flexibility, which when considering an FPGA design framework, turns into a barrier to leverage the energy and throughput benefits of specialization, making current ISA-based architectures bad contenders against GPUs for the task of interest.

Current FPGA accelerators may be not suitable for multi-tenancy serving systems because ISA-based architectures are not efficient, and Template-based rely on device reconfiguration which is too slow for our use case, but this does not constitute a fundamental limitation for FPGA use in model serving systems, especially when considering a more limited selection of models, where some degree of specialization of the accelerator would be feasible.

5.3. Future plans

As mentioned in the literature review. FPGAs are being successfully employed for more specific serving cases. By changing the use case we studied in this work, we believe it is possible to be able to serve a variety of models without device programming or the data moving burden of an ISA-based accelerator. Current industry standard DNN models feature scores of layer sizes and different computing requirements, but the same does not hold between families of models. We think that the efficient and deterministic Template-based approach can be extended to support such families, such as the Resnet family, by designing a layer accelerator approach to efficiently execute common layers between models. This approach could bring Template-based accelerators to a position where they are able to support more models, on a scheme that may allow multi-tenancy of highly efficient accelerators, under a more limited model support set.

Bibliography

[1] K. Hazelwood, S. Bird, D. Brooks, S. Chintala, U. Diril, D. Dzhulgakov, M. Fawzy, B. Jia, Y. Jia, A. Kalro, *et al.*, "Applied Machine Learning at Facebook: A Datacenter Infrastructure Perspective," in *Proceedings of the 24th IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2018.

[2] C.-J. Wu, D. Brooks, K. Chen, D. Chen, S. Choudhury, M. Dukhan, K. Hazelwood, E. Isaac, Y. Jia, B. Jia, *et al.*, "Machine Learning at Facebook: Understanding Inference at the Edge," in *Proceedings of the 2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2019.

[3] D. Crankshaw, X. Wang, G. Zhou, M. J. Franklin, J. E. Gonzalez, and I. Stoica, "Clipper: A low-latency online prediction serving system," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, Boston, MA: USENIX Association, Mar. 2017, pp. 613–627, ISBN: 978-1-931971-37-9. [Online]. Available: `https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/crankshaw`.

[4] *Google AI Platform*, Retrieved May 2020 from `https://cloud.google.com/ai-platform/`, 2020.

[5] C. Olston, N. Fiedel, K. Gorovoy, J. Harmsen, L. Lao, F. Li, V. Rajashekhar, S. Ramesh, and J. Soyke, "TensorFlow-Serving: Flexible, High-Performance ML Serving," *Workshop on ML Systems at NeurIPS 2017*, 2017.

[6] F. Romero, Q. Li, N. J. Yadwadkar, and C. Kozyrakis, *Infaas: A model-less and managed inference serving system*, 2020. arXiv: `1905.13348 [cs.DC]`.

[7] P. Mattson, V. J. Reddi, C. Cheng, C. Coleman, G. Diamos, D. Kanter, P. Micikevicius, D. Patterson, G. Schmuelling, H. Tang, *et al.*, "MLPerf: An industry standard benchmark suite for machine learning performance," *IEEE Micro*, vol. 40, no. 2, pp. 8–16, 2020.

[8] A. Gujarati, R. Karimi, S. Alzayat, A. Kaufmann, Y. Vigfusson, and J. Mace, "Serving dnns like clockwork: Performance predictability from the bottom up," *CoRR*, vol. abs/2006.02464,

2020. arXiv: 2006.02464. [Online]. Available: https://arxiv.org/abs/2006.02464.

[9] *Amazon web services*. [Online]. Available: https://aws.amazon.com/.

[10] *Google cloud*. [Online]. Available: https://cloud.google.com/.

[11] "Alibaba fpga cloud instances," [Online]. Available: https://www.alibabacloud.com/help/doc-detail/108504.htm.

[12] Y. Chen, A. Das, W. Qin, A. Sivasubramaniam, Q. Wang, and N. Gautam, "Managing server energy and operational costs in hosting centers," ser. SIGMETRICS '05, Banff, Alberta, Canada: Association for Computing Machinery, 2005, pp. 303–314, ISBN: 1595930221. DOI: 10.1145/1064212.1064253. [Online]. Available: https://doi.org/10.1145/1064212.1064253.

[13] V. Sze, Y.-H. Chen, J. Emer, A. Suleiman, and Z. Zhang, "Hardware for machine learning: Challenges and opportunities," *2017 IEEE Custom Integrated Circuits Conference (CICC)*, Apr. 2017. DOI: 10.1109/cicc.2017.7993626. [Online]. Available: http://dx.doi.org/10.1109/CICC.2017.7993626.

[14] "Amazon fpga web services," [Online]. Available: https://aws.amazon.com/ec2/instance-types/f1/.

[15] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P.-l. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon, "In-datacenter performance analysis of a tensor processing unit," in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, 2017, pp. 1–12. DOI: 10.1145/3079856.3080246.

[16] "Accelerating facebook's infrastructure with application-specific hardware," [Online]. Available: https://engineering.fb.com/2019/03/14/data-center-engineering/accelerating-infrastructure/.

[17] K. Guo, S. Zeng, J. Yu, Y. Wang, and H. Yang, "[dl] a survey of fpga-based neural network inference accelerators," vol. 12, no. 1, Mar. 2019, ISSN: 1936-7406. DOI: 10.1145/3289185. [Online]. Available: https://doi.org/10.1145/3289185.

[18] S. Mavridis, M. Pavlidakis, I. Stamoulias, C. Kozanitis, N. Chrysos, C. Kachris, D. Soudris, and A. Bilas, "Vinetalk: Simplifying software access and sharing of fpgas in datacenters," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, 2017, pp. 1–4. DOI: 10.23919/FPL.2017.8056788.

[19] J. Guo, H. He, T. He, L. Lausen, M. Li, H. Lin, X. Shi, C. Wang, J. Xie, S. Zha, *et al.*, "GluonCV and GluonNLP: Deep Learning in Computer Vision and Natural Language Processing," *Journal of Machine Learning Research*, vol. 21, no. 23, pp. 1–7, 2020.

[20] H. Zhang, C. Wu, Z. Zhang, Y. Zhu, Z. Zhang, H. Lin, Y. Sun, T. He, J. Mueller, R. Manmatha, *et al.*, "ResNeSt: Split-Attention Networks," *arXiv preprint arXiv:2004.08955*, 2020.

[21] O. B. Sezer and A. M. Ozbayoglu, "Algorithmic financial trading with deep convolutional neural networks: Time series to image conversion approach," *Applied Soft Computing*, vol. 70, pp. 525–538, 2018, ISSN: 1568-4946. DOI: https://doi.org/10.1016/j.asoc.2018.04.024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1568494618302151.

[22] D. Agarwal, B. Long, J. Traupman, D. Xin, and L. Zhang, "LASER: A Scalable Response Prediction Platform for Online Advertising," in *Proceedings of the 7th ACM International Conference on Web Search and Data Mining (WSDM)*, 2014.

[23] B. Dalessandro, D. Chen, T. Raeder, C. Perlich, M. Han Williams, and F. Provost, "Scalable Hands-Free Transfer Learning for Online Advertising," in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2014.

[24] G. Campagna, R. Ramesh, S. Xu, M. Fischer, and M. S. Lam, "Almond: The Architecture of an Open, Crowdsourced, Privacy-Preserving, Programmable Virtual Assistant," in *Proceedings of the 26th International World Wide Web Conference (WWW)*, 2017.

[25] K. Sokol and P. A. Flach, "Glass-Box: Explaining AI Decisions With Counterfactual Statements Through Conversation With a Voice-enabled Virtual Assistant," in *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI)*, 2018.

[26] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[27] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016, pp. 367–379. DOI: 10.1109/ISCA.2016.40.

[28] J. Fowers, K. Ovtcharov, M. Papamichael, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman, L. Adams, M. Ghandi, S. Heil, P. Patel, A. Sapek, G. Weisz, L. Woods, S. Lanka, S. K. Reinhardt, A. M. Caulfield, E. S. Chung, and D. Burger, "A configurable cloud-scale dnn processor for real-time ai," in *Proceedings of the 45th Annual International Symposium on Computer Architecture*, ser. ISCA '18, Los Angeles, California: IEEE Press, 2018, pp. 1–14, ISBN: 9781538659847. DOI: 10.1109/ISCA.2018.00012. [Online]. Available: https://doi.org/10.1109/ISCA.2018.00012.

[29] "Nvidia t4: Tensor core gpu for ai inference.," [Online]. Available: https://www.nvidia.com/en-us/data-center/tesla-t4/.

[30] G. Nguyen, S. Dlugolinsky, M. Bobak, V. Tran, A. Lopez Garcia, I. Heredia, P. Malík, and L. Hluchý, "Machine learning and deep learning frameworks and libraries for large-scale data mining: A survey," *Artificial Intelligence Review*, vol. 52, pp. 77–124, Jun. 2019. DOI: 10.1007/s10462-018-09679-z.

[31] *Open Neural Network Exchange Format: The new open ecosystem for interchangeable AI models*, Retrieved May 2020 from https://onnx.ai/, 2020.

[32] *Neural Network Exchange Format (NNEF)*, Retrieved May 2020 from https://www.khronos.org/nnef/, 2020.

[33] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, H. Shen, M. Cowan, L. Wang, Y. Hu, L. Ceze, *et al.*, "TVM: An Automated End-to-End Optimizing Compiler for Deep Learning," in *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2018.

[34] "Xla: Optimizing compiler for machine learning," [Online]. Available: https://www.tensorflow.org/xla.

[35] K. Hazelwood, S. Bird, D. Brooks, S. Chintala, U. Diril, D. Dzhulgakov, M. Fawzy, B. Jia, Y. Jia, A. Kalro, *et al.*, "Applied Machine Learning at Facebook: A Datacenter Infrastructure Perspective," in *Proceedings of the 24th IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2018.

[36] J. Soifer, J. Li, M. Li, J. Zhu, Y. Li, Y. He, E. Zheng, A. Oltean, M. Mosyak, C. Barnes, T. Liu, and J. Wang, "Deep learning inference service at microsoft," in *2019 USENIX Conference on Operational Machine Learning (OpML 19)*, Santa Clara, CA: USENIX Association, May 2019, pp. 15–17, ISBN: 978-1-939133-00-7. [Online]. Available: https://www.usenix.org/conference/opml19/presentation/soifer.

[37] A. Samanta, S. Shrinivasan, A. Kaufmann, and J. Mace, *No dnn left behind: Improving inference in the cloud with multi-tenancy*, 2019. arXiv: 1901.06887 [cs.DC].

[38] S. K. Zaman, A. u. R. Khan, J. Shuja, T. Maqsood, S. Mustafa, and F. Rehman, "A systems overview of commercial data centers: Initial energy and cost analysis," *International Journal of Information Technology and Web Engineering*, vol. 14, pp. 42–65, Jan. 2019. DOI: 10.4018/IJITWE.2019010103.

[39] H. Zhang, G. Ananthanarayanan, P. Bodik, M. Philipose, P. Bahl, and M. J. Freedman, "Live video analytics at scale with approximation and delay-tolerance," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, Boston, MA: USENIX Association, Mar. 2017, pp. 377–392, ISBN: 978-1-931971-37-9. [Online]. Available: https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/zhang.

[40] V. Stantchev and C. Schröpfer, "Negotiating and enforcing qos and slas in grid and cloud computing," May 2009, pp. 25–35, ISBN: 978-3-642-01670-7. DOI: 10.1007/978-3-642-01671-4_3.

[41] W. Chee Yau, *How Zendesk Serves TensorFlow Models in Production*, https://medium.com/zendesk-engineering/how-zendesk-serves-tensorflow-models-in-production-751ee22f0f4b, Feb. 2017.

[42] J. Hermann and M. Del Balso, *Scaling Machine Learning at Uber with Michelangelo*, https://eng.uber.com/scaling-michelangelo/, Nov. 2018.

[43] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, *et al.*, "In-Datacenter Performance Analysis of a Tensor Processing Unit," in *Proceedings of the 44th ACM/IEEE International Symposium on Computer Architecture (ISCA)*, 2017.

[44] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[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, Apr. 2009, ISSN: 0001-0782. DOI: 10.1145/1498765.1498785. [Online]. Available: https://doi.org/10.1145/1498765.1498785.

[46] S. Wang, A. Pathania, and T. Mitra, "Neural network inference on mobile socs," *IEEE Design and Test*, vol. 37, no. 5, pp. 50–57, Oct. 2020, ISSN: 2168-2364. DOI: 10.1109/mdat.2020.2968258. [Online]. Available: http://dx.doi.org/10.1109/MDAT.2020.2968258.

[47] R. Dennard, F. Gaensslen, H.-N. Yu, V. Rideout, E. Bassous, and A. LeBlanc, "Design of ion-implanted mosfet's with very small physical dimensions," *IEEE Journal of Solid-State Circuits*, vol. 9, no. 5, pp. 256–268, 1974. DOI: 10.1109/JSSC.1974.1050511.

[48] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *2011 38th Annual International Symposium on Computer Architecture (ISCA)*, 2011, pp. 365–376.

[49] R. Hameed, W. Qadeer, M. Wachs, O. Azizi, A. Solomatnikov, B. C. Lee, S. Richardson, C. Kozyrakis, and M. Horowitz, "Understanding sources of inefficiency in general-purpose chips," *SIGARCH Comput. Archit. News*, vol. 38, no. 3, pp. 37–47, Jun. 2010, ISSN: 0163-5964. DOI: 10.1145/1816038.1815968. [Online]. Available: https://doi.org/10.1145/1816038.1815968.

[50] "Nvidia tesla v100 architecture," [Online]. Available: https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf.

[51] R. K. Sharma and M. Casas, "Wavefront parallelization of recurrent neural networks on multi-core architectures," in *Proceedings of the 34th ACM International Conference on Supercomputing*, ser. ICS '20, Barcelona, Spain: Association for Computing Machinery, 2020, ISBN: 9781450379830. DOI: 10.1145/3392717.3392762. [Online]. Available: https://doi.org/10.1145/3392717.3392762.

[52] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam, "Shidiannao: Shifting vision processing closer to the sensor," in *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, 2015, pp. 92–104. DOI: 10.1145/2749469.2750389.

[53] Y. E. Wang, G.-Y. Wei, and D. Brooks, *Benchmarking tpu, gpu, and cpu platforms for deep learning*, 2019. arXiv: 1907.10701 [cs.LG].

[54] Z. Wen, J. Shi, B. He, J. Chen, K. Ramamohanarao, and Q. Li, "Exploiting gpus for efficient gradient boosting decision tree training," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 12, pp. 2706–2717, 2019. DOI: 10.1109/TPDS.2019.2920131.

[55] S. Windh, X. Ma, R. J. Halstead, P. Budhkar, Z. Luna, O. Hussaini, and W. A. Najjar, "High-level language tools for reconfigurable computing," *Proceedings of the IEEE*, vol. 103, no. 3, pp. 390–408, 2015. DOI: 10.1109/JPROC.2015.2399275.

[56] J. Bolaria, "Soc fpgas fuel next generation of iot, data center and communications infrastructure applications through power efficient processing," [Online]. Available: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/third-party/asdf-linley-soc-fpgas-fuel-next-generation.pdf.

[57] "How fpgas accelerate financial services workloads," [Online]. Available: https://www.hpcwire.com/2018/09/11/how-fpgas-accelerate-financial-services-workloads/.

[58] L. Wu, D. Bruns-Smith, F. A. Nothaft, Q. Huang, S. Karandikar, J. Le, A. Lin, H. Mao, B. Sweeney, K. Asanović, D. A. Patterson, and A. D. Joseph, "Fpga accelerated indel realignment in the cloud," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2019, pp. 277–290. DOI: 10.1109/HPCA.2019.00044.

[59] A. Vaishnav, K. D. Pham, J. Powell, and D. Koch, "Fos: A modular fpga operating system for dynamic workloads," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 13, no. 4, Sep. 2020, ISSN: 1936-7406. DOI: 10.1145/3405794. [Online]. Available: https://doi.org/10.1145/3405794.

[60] S. Mavridis, M. Pavlidakis, I. Stamoulias, C. Kozanitis, N. Chrysos, C. Kachris, D. Soudris, and A. Bilas, "Vinetalk: Simplifying software access and sharing of fpgas in datacenters," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, 2017, pp. 1–4. DOI: 10.23919/FPL.2017.8056788.

[61] "Ultrascale architecture dsp slice," [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug579-ultrascale-dsp.pdf.

[62] "Intel stratix 10 variable precision dsp blocks user guide," [Online]. Available: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/stratix-10/ug-s10-dsp.pdf.

[63] S. Bianco, R. Cadene, L. Celona, and P. Napoletano, "Benchmark analysis of representative deep neural network architectures," *IEEE Access*, vol. 6, pp. 64 270–64 277, 2018, ISSN:

2169-3536. DOI: 10.1109/access.2018.2877890. [Online]. Available: http://dx.doi.org/10.1109/ACCESS.2018.2877890.

[64] "Zynq dpu v3.3 product guide," [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/dpu/v3_3/pg338-dpu.pdf.

[65] "Deep learning with int8 optimization on xilinx devices," [Online]. Available: https://www.xilinx.com/support/documentation/white_papers/wp486-deep-learning-int8.pdf.

[66] N. Brown and D. Dolman, "It's all about data movement: Optimising fpga data access to boost performance," in *2019 IEEE/ACM International Workshop on Heterogeneous High-performance Reconfigurable Computing (H2RC)*, 2019, pp. 1–10. DOI: 10.1109/H2RC49586.2019.00006.

[67] H. Ye, X. Zhang, Z. Huang, G. Chen, and D. Chen, *Hybriddnn: A framework for high-performance hybrid dnn accelerator design and implementation*, 2020. arXiv: 2004.03804 [cs.AR].

[68] X. Zhang, J. Wang, C. Zhu, Y. Lin, J. Xiong, W.-m. Hwu, and D. Chen, "Dnnbuilder: An automated tool for building high-performance dnn hardware accelerators for fpgas," in *Proceedings of the International Conference on Computer-Aided Design*, ser. ICCAD '18, San Diego, California: Association for Computing Machinery, 2018, ISBN: 9781450359504. DOI: 10.1145/3240765.3240801. [Online]. Available: https://doi.org/10.1145/3240765.3240801.

[69] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "Finn," *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, Feb. 2017. DOI: 10.1145/3020078.3021744. [Online]. Available: http://dx.doi.org/10.1145/3020078.3021744.

[70] T. Moreau, T. Chen, L. Vega, J. Roesch, E. Yan, L. Zheng, J. Fromm, Z. Jiang, L. Ceze, C. Guestrin, and A. Krishnamurthy, *A hardware-software blueprint for flexible deep learning specialization*, 2019. arXiv: 1807.04188 [cs.LG].

[71] Y. Chen, J. He, X. Zhang, C. Hao, and D. Chen, "Cloud-dnn: An open framework for mapping dnn models to cloud fpgas," in *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '19, Seaside, CA, USA: Association for Computing Machinery, 2019, pp. 73–82, ISBN: 9781450361378. DOI: 10.1145/3289602.3293915. [Online]. Available: https://doi.org/10.1145/3289602.3293915.

[72] H. Sharma, J. Park, D. Mahajan, E. Amaro, J. K. Kim, C. Shao, A. Mishra, and H. Esmaeilzadeh, "From high-level deep neural models to fpgas," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2016, pp. 1–12. DOI: 10.1109/MICRO.2016.7783720.

[73] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, *Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1*, 2016. arXiv: 1602.02830 [cs.LG].

[74] "Xilinx large fpga methodology guide," [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx14_4/ug872_largefpga.pdf.

[75] *Vitis-ai: Adaptable and real-time ai inference acceleration.* [Online]. Available: https://www.xilinx.com/products/design-tools/vitis/vitis-ai.html.

[76] J. Aycock, "A brief history of just-in-time," *ACM Comput. Surv.*, vol. 35, no. 2, pp. 97–113, Jun. 2003, ISSN: 0360-0300. DOI: 10.1145/857076.857077. [Online]. Available: https://doi.org/10.1145/857076.857077.

[77] R. V. Chakaravarthy and H. Jiang, "Special session: Xta: Open source extensible, scalable and adaptable tensor architecture for ai acceleration," in *2020 IEEE 38th International Conference on Computer Design (ICCD)*, 2020, pp. 53–56. DOI: 10.1109/ICCD50377.2020.00026.

[78] "Axi reference guide," [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/axi_ref_guide/latest/ug1037-vivado-axi-reference-guide.pdf.

[79] *Zynq ultrascale mpsoc cache coherency.* [Online]. Available: https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842098/Zynq+UltraScale+MPSoC+Cache+Coherency.

[80] Y. Li, R. Gong, X. Tan, Y. Yang, P. Hu, Q. Zhang, F. Yu, W. Wang, and S. Gu, *Brecq: Pushing the limit of post-training quantization by block reconstruction*, 2021. arXiv: 2102.05426 [cs.LG].

[81] *Pre-tuned autotvm configurations (parameters) for common neural networks.* [Online]. Available: https://github.com/tlc-pack/tophub.

[82] *Traffic shaping of hp ports on zynq ultrascale+.* [Online]. Available: https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/688128001/.

[83] *Xilinx multi-node technology leadership continues with ultrascale+ portfolio "3d on 3d" solutions*. [Online]. Available: https://www.xilinx.com/support/documentation/white_papers/wp472-3D-on-3D.pdf.

[84] *Ultra96 fpga soc single board computer*. [Online]. Available: https://www.avnet.com/wps/portal/us/products/avnet-boards/avnet-board-families/ultra96-v2/.

[85] *Python productivity for zynq (pynq)*. [Online]. Available: https://pynq.readthedocs.io/en/latest/index.html.

[86] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, 2015. arXiv: 1512.03385 [cs.CV].

[87] *Gluon cv toolkit*. [Online]. Available: https://cv.gluon.ai/.

[88] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.

[89] *Sysbench, scriptable multi-threaded benchmark tool*. [Online]. Available: https://github.com/akopytov/sysbench.

[90] *Power management bus*. [Online]. Available: https://pmbus.org/.

[91] *Ultra96v2 schematics*. [Online]. Available: https://www.96boards.org/documentation/consumer/ultra96/ultra96-v2/hardware-docs/files/ultra96-v2-schematics.PDF.

[92] *Nvidia system management interface*. [Online]. Available: https://developer.nvidia.com/nvidia-system-management-interface.

[93] *Breathe new life into your data center with alveo adaptable accelerator cards*. [Online]. Available: https://www.xilinx.com/support/documentation/white_papers/wp499-alveo-intro.pdf.

[94] *Gpus at google cloud*. [Online]. Available: https://cloud.google.com/gpu.