Performance-aware Coarse-Grained Reconfigurable Logic Accelerator for Deep Learning Applications

A Thesis submitted in partial fulfillment of the requirements for the degree of Master of Science at George Mason University

by

Katherine Mercado Rejas
Bachelor's in Electronics and Communication Engineering
Polytechnic University of Madrid, 2018

Director: Sai Manoj Pudukotai Dinakarrao, Assistant Professor
George Mason University

Summer Semester 2022
George Mason University
Fairfax, VA

DEDICATION

This is dedicated to my grandmother, Consuelo.

ACKNOWLEDGEMENTS

I would like to thank my relatives and supporters who have made this happen. Dr. Carlos Carreras for his assistantship with his writing skills and notes. Dr. Sai Manoj for his guidance and wisdom during the whole process of my thesis and the other members of my committee who were of invaluable help. Finally, thanks go out to the staff from the department who provided me with assistance during my whole master's degree. Special thanks to Dr. Dimitrios Ioannou, who supported me with advising during my first year.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF EQUATIONS

LIST OF ABBREVIATIONS

Deep Neural Network ..... DNN
Convolutional Neural Network ..... CNN
Processing Element ..... PE
Coarse-Grained Reconfigurable Architecture ..... CGRA
Graphical processing unit ..... GPU
General Purpose Graphical processing unit ..... GPGPU
Central processing unit ..... CPU
Application-Specific Integration Circuit ..... ASIC
Artificial Intelligence ..... AI
Compute Unified Device Architecture ..... CUDA
Open Computing Language ..... OpenCL
Look Up Table ..... LUT
Programmable Logic Devices ..... PLD

ABSTRACT

PERFORMANCE-AWARE COARSE-GRAINED RECONFIGURABLE LOGIC ACCELERATOR FOR DEEP LEARNING APPLICATIONS

Katherine Mercado Rejas, M.S.

George Mason University, 2022

Thesis Director: Dr. Sai Manoj Pudukotai Dinakarrao

Deep neural networks (DNNs) are widely deployed in various cognitive applications including computer vision, speech recognition, and image processing. The surpassing accuracy and performance of deep neural networks come at the cost of high computational complexity. Therefore, software implementations of DNNs and convolutional neural networks (CNNs) are often hindered by computational and communication bottlenecks. As a panacea, numerous hardware accelerators are introduced in recent times to accelerate DNNs and CNNs. Despite effectiveness, the existing hardware accelerators are often confronted by the involved computational complexity and the need for special hardware units to implement each of the DNN/CNN operations. To address such challenges, a reconfigurable DNN/CNN accelerator is proposed in this work. The proposed architecture comprises nine processing elements (PEs) that can perform both convolution and arithmetic operations through run-time

reconfiguration and with minimal overhead. To reduce the computational complexity, we employ Mitchell's algorithm, which is supported through low-overhead coarse-grained reconfigurability in this work. To facilitate efficient data flow across the PEs, we pre-compute the dataflow paths and configure the dataflow during the runtime. The proposed design is realized on a field-programmable gate array (FPGA) platform for evaluation.

INTRODUCTION

Deep learning algorithms including Deep neural networks (DNNs) and convolutional neural networks (CNNs) have been widely adopted in a plethora of applications in recent times. These techniques exploit the internal correlation between the data samples in each of the internal hidden layers to perform the tasks such as classification or prediction with high accuracy. Therefore, it excels at complex image classification, natural language processing, computer vision, and speed recognition problems among others.

As a result of these characteristics, deep learning techniques require high-end infrastructure due to the large number of parameters that will need to be trained within an acceptable amount of time, compared to traditional machine learning algorithms. Besides, operations can be computed faster in hardware that has been customized for that particular application. Therefore, hardware acceleration refers to customizable hardware for applications that demand high parallelism and reduced overhead.

Implementing ML and other learning methodologies on traditional CPUs is facing a formidable challenge in terms of inference latency, memory accesses, and energy efficiency due to the lack of temporal data locality and logic-memory communication Graphics processing units (GPUs) and custom-designed accelerators (ASICs) [1] are designed for enhanced hardware performance. However, the performance and efficiency

of such architectural paradigms are limited due to power consumption, costs, and reconfigurability.

GPU platforms

Graphical processing units and central processing units progress towards becoming fundamental hardware to deploy DNNs. However, despite the versatility that the latter offers for large-scale data applications, the performance may produce an unsatisfactory result. This stems from the fact that the CPUs cores may wait to be stuck on memory for intensive memory applications, hence degrading its parallel computing performance.

On the other hand, the advantage of GPUs over CPUs greatly benefits a large number of computations. Because thread parallelism hides latency, it offers a high bandwidth for substantial chunks of memory as well as high throughput. To achieve high-performance GPUs due to the emergence of deep learning applications, some platforms such as CUDA or OpenCL have been available to create modern GPU accelerators for general purpose computations (GPGPU). Moreover, it can be found some deep learning GPU-based libraries that assist these implementations such as cuDNN [2] and Cuda-convnet [3]. Due to the superior parallel processing performance and their floating-point characteristics, GPUs have undergone significant development in recent years. Figure 1 illustrates the performance of Nvidia GPUs in the number of floating-point operations per second, GFLOPs and power consumption, and TPD from 2006 to 2018.

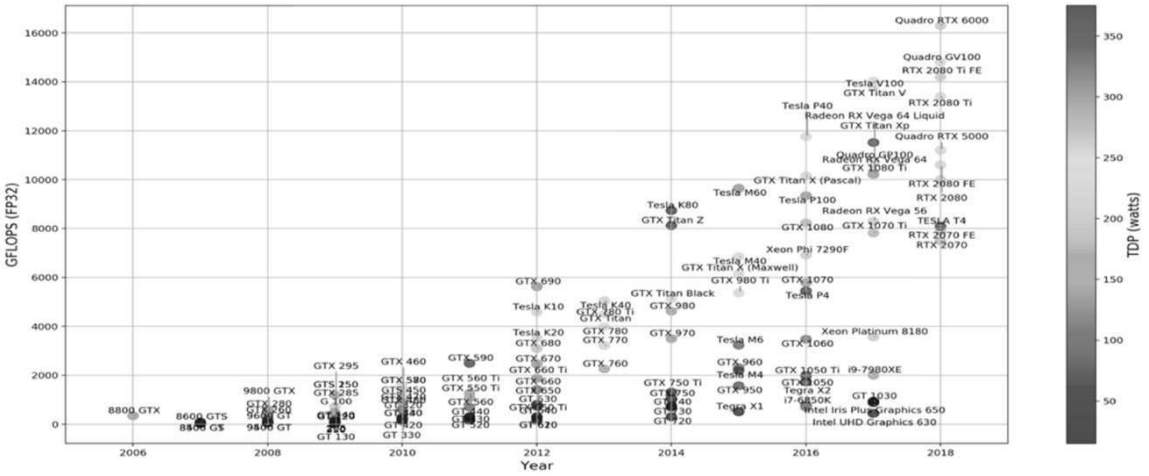

Figure 1 GPU throughput in floating point operations per second and power consumption.

However, GPUs require high power consumption and cost and, as a result, alternative hardware solutions are needed to alleviate power consumption while improving computing performance. An alternative solution is FPGA platforms.

FPGA platforms

Field programmable gate array (FPGA)-based implementations are adopted recently for the deployment of numerous applications including DNNs and CNNs due to their programmability, reconfigurability, and flexibility. Logic density on the state-of-the-art FPGAs allows good performance for these intensive computations. Another major advantage is FPGA's support for fine-grained and bit-level operations when compared to GPUs and ASICs makes this platform draw attention to low latency applications. Numerous FPGA-based DL accelerators are proposed in the literature [4].

A significant difference in utilization between FPGAs and GPUs is that the former is normally used for DNN inference while the latter is for DNN training. This is

because the memory bandwidth found in FPGAs is much lower than the memory bandwidth found in GPUs [5]. To overcome this major disadvantage, there are three different algorithm optimizations that address this issue. Firstly, algorithm operation, such as the fast Fourier transform, helps to reduce the number of arithmetic operations during the inference performance. Secondly, the data-path optimization consists of applying the *unrolling* technique to the convolutional layers for parallelism. Lastly, model compression consists of eliminating the redundant parameters in error-tolerant applications such as *pruning* technique. Figure 2 illustrates the performance of a few CNN models on FPGA devices.

Table 1 CNN models performance on FPGA platform

CNN Model	FPGA Device	Optimization Method	Accuracy (Top-5)	# of Param (M)	Computation (GOP)	Precision	Frequency (MHz)	Throughput (GOP/s)	Power (W)
VGG-19 [139]	Arria10 GX1150	–	90.1%	138	30.8	float32	370	866	41.7
VGG-16 [114]	Zynq 7Z045	SVD	87.96%	50.2	30.5	fixed16	150	137	9.6
VGG-16 [134]	Arria10 GX1150	Dynamic	88.1%	138	30.8	fixed8	150	645	–
BNN: XNOR-Net [137]	Stratix5 GSD8	Binary	66.8%	87.1	2.3	fixed1	150	1,964	26.2
Ternary ResNet [115]	Stratix10	Ternary, pruning	79.7%	61	1.4	float32	500	12,000	141.2

FPGA platforms though can enable reconfigurability and programmability, still incurs large resource utilization and latency when deployed for DL applications. To partially address this, the existing works have exploited the resilience of DNNs and CNNs despite utilizing low-precision data. In addition, approximate computing has been enabled to address the latency challenges [6].

In the literature, approximate arithmetic units such as dividers [7], adders [8], and multipliers [9] have been developed to implement DNN and CNN operations. The

majority of these architectures are pre-configured for application-specific designs, confining their applicability to a specific architecture. Thus, the required resources and computational complexity can be reduced. However, the challenges of reconfiguration overheads and the computational complexity to perform MAC operations still remain unanswered.

In contrast, the reconfigurability of the FPGA architectures, arithmetics behind the approximations, and MAC computations are exploited to propose the coarse-grained reconfigurable high-speed approximate accelerator for DL applications. For this purpose, a 3x3 tile structure of processing elements (PEs) is designed, and reconfigurable through programming words to perform a wide variety of operations including add, subtract, multiply, divide, logarithm, and anti-logarithm. Mitchell algorithm [10] is employed to reduce the complexity of the resource-intensive multiply operations and enhance the involved computational latency. The high-speed reconfigurability and interconnectivity of FPGAs make the proposed design of PEs energy-efficient and reconfigurable with minimal overheads.

The proposed architecture is also designed to facilitate programmability, reconfigurability, and applicability to other applications with minimal overheads. The novel contributions of this work can be outlined in a three-fold manner as follows:

- Mitchell algorithm-inspired reconfigurable PEs are designed to perform the MAC operations for DNNs and CNNs. The proposed design employs 8-bit operands to further minimize the computational overheads without impacting the performance of DNN/CNN implementation.

- A coarse-grained reconfigurable DNN/CNN accelerator on the FPGA platform is proposed to minimize the reconfiguration overheads and enable adaptability to a wide range of applications.
- A weight-stationary approach is employed for seamless dataflow across the PE cores in the proposed architecture. Look-up-Table-based log and antilog blocks are designed to support Mitchell's algorithm-based approximate multipliers with low latency.

The proposed coarse-grained reconfigurable architecture is evaluated on Versal VCK190 FPGA platform for DNN and CNN networks. The proposed accelerator is at least 1.25x faster than previous work [11] with 8-bit precision and 1.87x faster than [12], [13] with 16 and 32-bits respectively. Moreover, it shows improvement in both area and energy.

REVIEW OF LITERATURE

Deep learning techniques including DNNs and CNNs compose millions of MAC operations. These operations can be performed in a parallel manner. As such, FPGA platforms are one of the best-suited platforms for DL acceleration by exploiting their inherent parallelism. Numerous works have proposed FPGA-based accelerators for DL applications [11], [12], [13]. The great challenge for a hardware accelerator design is to find the best trade-off between power, performance, and reconfigurability.

Reconfigurability brings forth advantages, as having more functionality employing fewer resources, helps cost savings. Moreover, it may extend the useful life of hardware by updating its purpose and achieving faster development. Coarse-grained reconfigurable architecture is a common type of reconfigurable architecture based on functional units such as PEs, in a mesh-style network. This type of architecture may perform complex operations while providing low power consumption, less configuration, and routing overhead.

A key point of taking advantage of PEs is to avoid the combination of configurable logic blocks (CLBs) compared to pure FPGA-based hardware accelerators, ensuring a decrease in both areas and routing overhead, such as the DReAm [14] and MORA [15] architectures.

The present memory bottleneck found in FPGA-based hardware accelerators is localized in data movement, which could result in more energy consumption than the computation. One goal of this paper is to alleviate such concerns by implementing weight-stationary dataflow to maximize access to computation results from the different PEs and thus, minimize energy consumption. CGRA-based architectures such as MORA [15], employ pipelined computational dataflow organized in two levels while eliminating the need for a centralized routing controller. Contrary to DreAm [14], which possesses a global unit for this purpose. Other works also, [16], [17], exploit the hierarchical dataflow concept for on-chip and inter-PE communication respectively, for both convolutional data and MAC operations. Another FPGA-based accelerator such as [18], proposed an adaptable reconfigurable datapath that allows depending on the operand, parallel or sequential dataflow for multiply operations.

FPGAs require a considerable amount of data reconfiguration for their programmable routing network. This is translated into a larger configuration time when multiple hardware configurations are involved in a single architecture. In order to overcome this, multiple-bit arithmetic processing elements, such as those working with 8 or 16 bits, as our proposed architecture, may be used where high efficiency is achieved for DNN/CNN computations while reducing power and area. Arithmetic hardware accelerator units are developed to carry neural network computations such as [11], [12], [13], multipliers and dividers are the most frequent types of independent arithmetic units. Few designs execute both operations where the lack of support for division may generate a large overhead in the design [19], [20]. The proposed architecture integrates flexibility

benefits that an FPGAs platform provides, as well as characteristics of a reconfigurable coarse-grained based processing element architecture for multiple hardware configurations.

PROPOSED APPROXIMATE COMPUTING-BASED HIGH-SPEED RECONFIGURABLE ACCELERATOR

The proposed high-speed reconfigurable approximate computing-based hardware accelerator architecture is presented in Figure 2. The proposed architecture comprises 9 processing elements (PEs), termed cores. Each of the PEs is designed to perform multiplications based on Mitchell's algorithm [10], discussed later. For the purpose of reconfigurability, the CGRA PEs are controlled using the codewords, defined through control bits (4-bits in our design). Depending on the control bits, the datapath and the PEs are configured. The reconfiguration time is reduced through the control words of the CGRA paradigm; therefore, it allows the usage of configuration memory more efficiently. The details of individual blocks are discussed below.

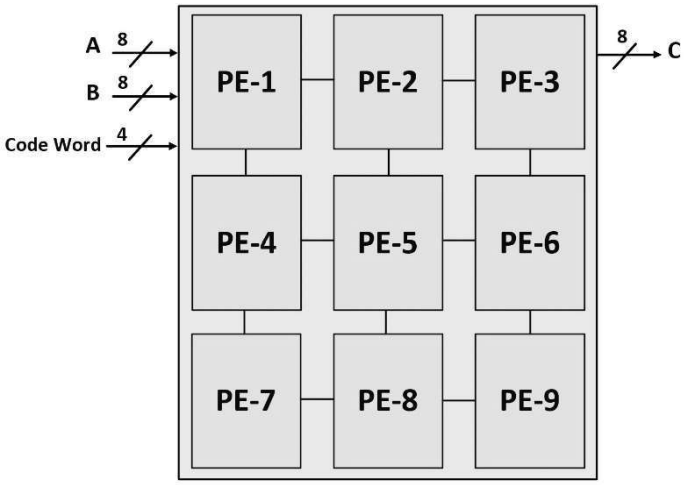

Figure 2 Proposed accelerator architecture showing 9 cores in a mesh-style network.

System Architecture

The overall structure of the proposed architecture is presented in Figure 2. It comprises nine PEs arranged in a tiled manner. In the highest level of description, this architecture considers two operands *A* and *B* of 8-bit precision along with a code word of 4-bits, referred to as *Mode* signal. The output of this PE is 8-bits. The codewords and the corresponding operation are represented in Table 2. In the current design, each operand is represented in a fixed-point format using 4-bits for the integer and 4-bits for the fractional part. The rationale to choose the fixed-point representation is its wide adoption for neural network applications, as it helps to reduce the logic usage and the power consumption on FPGAs platforms.

Table 2. Codewords and operations

	Logarithm Block	2's C. Block	Adder Block	Antilog Block
Adder			0001	
Subtraction		0010	0010	
Multiply	0100		0100	0100
Division	1000	1000	1000	1000

Processing Element Architecture

One of the common and computationally intensive operations in the DL applications as well as the generic application workloads is the multiply operation. The complexity further increases for fixed-point and floating-point data types. To address this challenge, we design our PE focusing on multiply operations. However, designing only

multiplier makes the design inefficient, as other operations such as additions and subtractions are critical in DL and generic workloads.

Considering the reconfigurability of the underlying FPGAs, we design the PE to support multiplications as well as other arithmetic operations in this project. For this purpose, we design the multiplier based on Mitchell's algorithm [10] as described below. As per Mitchell's algorithm, a multiply operation is defined as follows:

Equation 1 Multiplication

$$AB = 10^{\log(A)+\log(B)}$$

Equation 2 Division

$$\frac{A}{B} = 10^{\log(A)-\log(B)}$$

Where A and B are the operands. Based on equation 1 and equation 2, the PE architecture is designed as shown in Figure 3. Thus, the PE encompasses an adder, two's complement, log, and antilog blocks. Depending on the codeword, the individual units of the PE are enabled or disabled. For instance, to perform the multiplication operation, the log, adder, and antilog blocks will be enabled. Similarly, the codeword enables the log, adder, two's complement, and antilog blocks to perform the division operation. The proposed architecture thus is capable of performing the add, subtract, log, antilog, multiply, and divide operations in a seamless manner. This is facilitated through the reconfigurability of the underlying FPGA architecture. To perform the reconfigurability through the codewords in a CGRA manner, we design a finite estate machine (FSM). The

FSM is responsible to decode the codewords and enable the data flow in a dynamic manner. As shown in Table 2, depending on the codeword, the corresponding computational blocks in the PE will be enabled or disabled. Logarithm and antilogarithm functions are the complex functions to be designed through standard CMOS designs. For this purpose, we employ the Taylor's Series approximation and design the hardware accordingly.

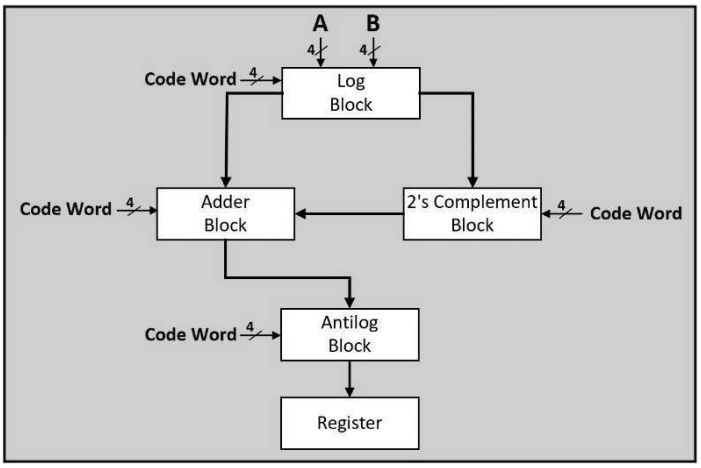

Figure 3 Processing element architecture showing the main four arithmetic blocks and register.

Taylor Series Algorithm for Log and Antilog operations

A Taylor Series is the expansion of a function into an infinite sum of terms. These terms are computed to get an approximate value for both logarithm and antilogarithm. The relationship between logarithm and antilogarithm to obtain a multiplication or a division is such that, the antilogarithm of the addition or subtraction of logarithms of *A* and *B* is the multiplication or the division of *A* and *B* respectively as shown in equation 1 and equation 2.

The logarithm function is achieved by the approximation of the Taylor Series for the natural logarithm and change of base property. The function *log(x)* is the approximation of the natural logarithm with *a* being the point where the function should be centered and *n* representing the number of terms as shown in equation 3.

Equation 3 Logarithm function

$$\log(x) = \frac{\sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{na^n}(x-a)^n}{ln10}$$

Equation 4 Multiply logarithm function

$$antilog(x) = \sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{na^n}(x-a)^n + \sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{na^n}(x-a)^n$$

To compute the antilogarithm for instance, for the multiply, the approximation between the logarithm and the exponential functions is made in equation 4. As observed from equation 3, direct implementation of logarithm and antilogarithm operations through CMOS design is inefficient due to the involved complexity. As such, we design a look-up-table (LUT)-based log and antilog units in this work. As FPGAs realize the design through LUTs, the design of logarithm and antilogarithm blocks are realized using the LUTs. This enables faster lookup, reduced computational complexity, and compatibility with the underlying FPGA architectures.

Application Mapping on the Proposed Accelerator

Convolutional layers are the building blocks for CNN, where convolution calculations are employed for feature extraction. An element-wise product is performed between an input tensor and a kernel array. After this, all terms are summed to obtain the value in that specific position of the output feature map [4]. The correspondent codewords for CNN computation active the PEs and the behavior involves further steps as shown in Figure 4.

Once the PEs have completed nine multiplications and have passed through the registers; these values are redirected back to the PEs to perform addition operations by utilizing and taking advantage of the same resources. Eight values are moved to the first four PEs, every two terms for each addition are designated with the same color label as illustrated in Figure 4. The result of these four additions is again redirected to the next PEs, activating the following two in the next cycle. In the third cycle, the two PEs results become the two next terms for the following addition. Finally, our last result can be sum to the result of the ninth PE which was unutilized in the first cycle.

This illustration demonstrates that the multiply calculation among the PEs will be executed in parallel independently of the number of inputs involved. After that, the time computation increases according to the number of additions that may be performed in each clock cycle. With different operations, when a PE is inactive, its output value is registered to be dispatched for immediate use while the remaining modules may start orchestrating other operations according to their codewords.

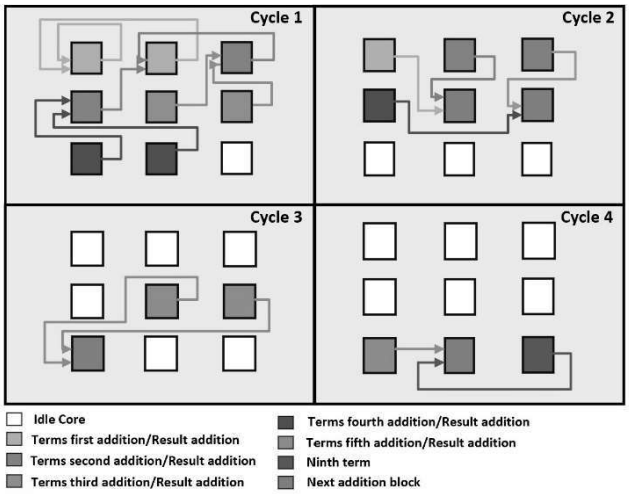

Figure 4 Sum of terms for a convolutional operation.

Applicability to Other Workloads

As aforementioned, one of the advantages of the proposed reconfigurable architecture is its applicability to a wide range of applications in addition to the DL acceleration. In order to employ this architecture for non-ML applications, the aforementioned application mapping needs to be reprogrammed. In other words, by redefining the associated data flow with each of the code words one can apply the proposed architecture to non-ML applications. For instance, instead of convolution operation, general matrix multiplication (GEMM) is a common operation encountered in a plethora of applications and workloads. To perform GEMM operation on the proposed architecture, the operand mapping and sum dataflow have to be redefined compared to the convolution operation. This can be performed in a non-complex manner through the programmability of the FPGAs. GEMM reuses the data during computation, therefore,

data movement and storage are drastically reduced, which leads to improved architectures.

RESULTS AND DISCUSSION

Simulation setup

The proposed architecture with nine processing element (PE) cores is implemented on Xilinx Versal VCK190 FPGA. The PEs are described using the very high-speed integrated circuit hardware description language (VHDL) and simulated using Modelsim Intel software. Further, it is synthesized as an IP block and verified in Xilinx Vivado 2021.1. The evaluation is performed in terms of the FPGA resources consumed for the deployment of the proposed architecture.

Evaluation of processing element

The data computation has been pipelined to reach the maximum possible frequency of operation. Furthermore, the scope of the data of both logarithm and antilogarithm computations is limited in the state machine to refine the code and reduce cycles. Area overhead has been decreased by reducing the precision of the data under performance constraints. Table 5 summarizes the proposed architecture results. Delay and resources have been obtained from Vivado synthesis. Power consumption is that of all the processing elements and their communication network.

Comparative performance

We compare the speed and the area of the arithmetic units from our proposed architecture with distinct arithmetic units as shown in Table 3. Many architectures found

in the literature are focused on a specific arithmetic computation [8], [9], [20], [21]. Moreover, there are works [22] that integrate adder and multiplier units in a single architecture. Recently, some interest has emerged in approximate multiplier and divider integrated architectures such as the tunable accuracy SIMDive architecture [23] using single instruction and multiple data, since this operation has great importance in some deep learning applications. As such, for a fair comparison, we configure the proposed architecture either as an adder, multiplier, or divider and compare it with the corresponding state-of-the-art works.

As can be seen from Table 3, the area increases for more complex computations such as multiplications and divisions. The proposed architecture shows area improvement in adder, multiplier and divider computation when compared to individual units [7], [8], [9], [19], [20], [21]. For instance, the best arithmetic unit implemented in [21] has more significant LUTs than the area of the proposed PE unit.

However, the proposed architecture has a wider benefit over existing architectures with multiple heterogeneous arithmetic blocks, as shown in Table 4. Observing a generic multiplier and adder architecture such as [22], the number of slices has to be multiplied by four. The architecture has been deployed using Virtex5, therefore, the number of LUTS increases until achieving a total of 4484 LUTs against 1023 found in the proposed architecture. Similar to the next case, for multiplier and subtractor units.

In Table 5 we compare our proposed architecture with previous FPGA-based accelerators of 8, 16 and 32 bits of precision.

Previous works such as [12], reduce the amount of off-chip data transfer by the optimization of its dataflow with an increment in BRAM usage, an improvement over the previous architecture [13]. This is demonstrated with a better power efficiency when compared to [11]. However, there is still power required for the off-chip memory. In our solution, we optimize power efficiency with optimized weight-stationary datapath, which decreases, even more, energy consumption.

Moreover, the proposed architecture is 1.24 times faster than [11] and 1.87 times faster than [12], [13]. Implementation has been optimized in power design and routing design, achieving lower use of resources for both LUTs and DSPs when compared to [11], [12], [13].

Table 3. Comparison of arithmetic units

Reference	Speed (Mhz)	Area (LUT)
Adder [8]	275	557
Adder [19]	71.017	932
Proposed Adder	187	243
Multiplier [20]	19.8	6971 Slices
Multiplier [9]	142.8	63400
Proposed Multiplier	187	780
Divider [7]	38	1060
Divider [21]	67.150	2472 Slices
Proposed Divider	187	815

Table 4. Comparison of arithmetic blocks

Reference	Speed (Mhz)	Area (LUT)
Adder + Multiplier [22]	415	1121 Slices
Proposed Adder + Multiplier	187	1023
Multiplier + Subtractor [22]	407	664 Slices
Proposed Multiplier + Subtractor	187	883

Table 5. FPGA-based accelerator comparison

	[11]	[12]	[13]	Proposed
FPGA	Startix-V	Arria-10	Zynq	Versal
Frequency (Mhz)	150	100	100	187
Precision	8-16bits	16-bits	32-bits	8bits
LUTs	161K	155K	118K	127K
DSPs	1518	784	824	196
Power (W)	21.1	9.4	9.4	8.7

CONCLUSION

In this work, a novel reconfigurable accelerator architecture aimed at CNN computations is presented. Its programmability allows it to perform addition, subtraction, multiplication, and division operations individually through a given mode. Its processing elements or cores are displayed following a coarse-grained reconfigurable architecture employing a weight-stationary approach as datapath. Each core can perform low precision operations of 8 bits word-length. Performance is evaluated with Versal VCK190. Compared with different arithmetic units aimed at CNN computations, the architecture shows improvement in the area, 1.22x lower resource utilization compared to the standard DNN/CNN accelerators. Further, it shows enhanced performance, is 1.24 times faster and 1.87 times faster than previous works.

REFERENCES

[1] Xin F., Youni J., Xuejiao Y., Ming D., Xin L. (2019). *Integration*. The VLSI Journal, Elsevier.

[2] Sharan C. (2014). *CUDNN: Efficient Primitives for Deep Learning*, arXiv: 1410.0759.

[3] Alex K. *Cudaconvet2*. Online. http://code.google.com/archive/p/cuda-convnet2/.

[4] Sathwika B., Abhijit D., Amlan G., Anand H., Hagar H., Sai M. (2022). *A Survey on Machine Learning Accelerators and Evolutionary Hardware Platforms*. IEEE Design Test 39, 3, 91-116.

[5] Feng, Xin and Jiang, Youni and Yang, Xuejiao and Du, Ming and Li, Xin. (2019). *Reconfigurable FPGA Architectures: Computer vision algorithms and hardware implementations: A survey.* Journal of Integration. 69, 309-320.

[6] Praveenjumar B., P. Eswaran. (2020). *Reconfigurable FPGA Architectures: A survey and Applications.* Journal of The Institution of Engineers.

[7] Gustavo S., Jean-Pierre D. (2009). *High Speed Fixed Point Dividers for FPGAs.* 448-452.

[8] Kapil Ram G., Neelam C., Sujeet M., Sandeep D. (2018). *A Parallel Pipelined Adder Suitable for FPGA Implementation.* 1-4.

[9] CRS Hanuman., J. Kamala. (2018). *Hardware implementation of 24-bit Vedic Multiplier in 32-bit floating-point divider.* 60-64

[10] Jhon M. (1962). *Computer Multiplication and Division Using Binary Logarithms.* IRE Transactions on Electronic Computers 4.

[11] Yufei M., Yu C., Sarma V., Jae-sun S. (2017). *Optimizing Loop Operation and Dataflow in FPGA Acceleration of Deep Convolutional Neural Networks.* 45-54.

[12] Qingchen X., Yun L., Liqiang L., Shengen Y., Yu-Wing T. (2017). *Exploring Heterogeneous Algorithms for Accelerating Deep Convolutional Neural Networks on FPGAs*. 1-6.

[13] Manoj A., Han C., Michael F., Peter M. (2016). *Fused-layer CNN Accelerators.* 1-2.

[14] Jurgen B., Thilo P., Christian H., Manfred G. (2001). *Design and Implementation of a Coarse-Grained Dynamically Reconfigurable Hardware Architecture.* 41-46.

[15] Marco L., Stefani P., Martin M., Pasquale C. (2005). *A New Reconfigurable Coarse-Grain Architecture for Multimedia Applications.* 119-126.

[16] Maurice P., Arnaud AA S., Bart M., Henk C. (2013). *Memory-Centric Accelerator Design for Convolutional Neural Networks.* IEEE 31st International Conference on Computer Design (ICCD).

[17] Yu-Hsin C., Tushar K., Joel S. E., Vivienne S. (2016). *Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks.* IEEE Journal of solid-state circuits 5,.127-138.

[18] Marco L., Stefani P., Martin M., Pasquale C. (2005). *Low-cost Fully Reconfigurable Data-path for FPGA-based Multimedia Processor. IEEE,* International Conference on Field Programmable Logic and Applications.

[19] Somsubhra G., Prarthana B., Arka D. (2013). *FPGA Based Implementation of a Double Precision IEEE Floating-Point Adder.* 271-275.

[20] Salty B., Sandip N. (2016). *VHDL Implementation of Self-Timed 32-bit Floating Point Multiplier with Carry Look Ahead Adder.* IEEE Journal of solid-state circuits 5,.127-138.

[21] Peter M. (2015). *High Throughput Floating-Point Dividers Implemented in FPGA.* 291-294.

[22] Lamiaa S., Abdel H., Khaled S., Hassan E., Mohamed E. (2020). *Design of Generic Floating-Point Multiplier and Adder/Subtractor Units.* 615-618.

[23] Zahra E., Salim H., Akash K. (2022). *SIMDive: Approximate SIMD Soft Multiplier-Divider for FPGAs with Tunable Accuracy.* 151-156.

BIOGRAPHY

Katherine Mercado Rejas received her Bachelor of Electronics and Communications engineering from the Polytechnic University of Madrid in 2018. She is employed as an intern in Volvo Trucks North America and she is doing her master's degree in Electrical Engineering at George Mason University.