Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 218 (2023) 469–478

Procedia Computer Science

www.elsevier.com/locate/procedia

International Conference on Machine Learning and Data Engineering

Customized FPGA Design and Analysis of Soft-Core Processor for DNN

Harini. Sriraman [a,*], Aswathy Ravikumar

[a]*School of Computer Science and Engineering, Vellore Institute of Technology, Chennai, India*

Abstract

Transistor scaling has continued to yield significant performance gains. The general-purpose architecture will not suffice in this dark silicon era with the big data boom and growing emphasis on performance. Artificial neural networks and deep learning are sophisticated computing activities that cannot be optimally executed (accuracy, performance, and energy) on today's general-purpose computer architecture. The demand for domain-specific computer architectures is increasing. This paper suggests and evaluates a highly reconfigurable domain-specific design for deep learning and artificial neural networks to overcome these problems. An FPGA-based soft-core processor for deep neural network (DNN) acceleration is suggested in this paper. Using a combination of unicast and multicast for collective operations, the proposed FPGA accelerates the performance of the deep learning network. The suggested design outperforms the current general-purpose processor by 6.6 times, according to a thorough examination and comparison. The existing architecture's level of accuracy is retained.

Keywords: Domain-Specific Architecture; Tensor Processing Unit; DNN; Deep learning; Customized FPGA

1. Introduction

Deep Neural Networks have been effectively used in a variety of domains, including object recognition [1], robotic[2], cybersecurity[3], medical diagnostics [4], and autonomous vehicles [5]. Typically, performance improvements are accompanied by the increased complexity of DNNs. Globally, a massive number of academic and industrial researchers are designing efficient hardware for the inference of DNNs Accelerators for DNNs are built with FPGAs, GPUs, and TPUs. GPUs provide massively parallel computational units and parallelize DNN calculations [6]. GPUs are energy, which restricts their use in embedded systems. FPGAs offer a high performance per watt and are

* Harini Sriraman
E-mail address: harini.s@vit.ac.in

1877-0509
10.1016/j.procs.2023.01.029

field customizable. FPGAs are frequently used for design prototyping and validation. ASICs are purpose-built for specific applications with optimal speed and power consumption.

The architecture of a computer system has always been significantly impacted by the underlying patterns and capabilities of hardware and software technologies [7], [8]. Programming improvements, such as moving from machine language to assembly language, from the prevalent programming paradigm of high-level procedural language to an object-oriented language. Failure of Moore's law: Gordon Moore projected in 1965 that the transistor count on a piece of silicon would double annually. Moore's law, now called, proved to be prophetic regarding the exponential expansion of computer power that led to the development of most of the contemporary world. Around 2010, though, Moore's Law started to fail, and many now wonder whether our era of tremendous growth is drawing to a close. Like the neuron in the brain, this web of transistors enables practically all modern gadgets to operate, from a digitized alarm to a supercomputer. The more transistor you can cram on a device, the more powerful this connection becomes computationally. For years, as semiconductors shrunk in size, their energy efficiency increased. They have, however, shrunk to the point where the channel that conducts the electrical charge through the transistor always can contain it. This creates heat, which may accelerate the wear and tear on the transistors, rendering them even more prone to leakage. However, heat is not restricted to a single transistor. Because the leakage of billions of transistors poses a significant danger to the chip's integrity, the processor must limit the amount of power it accepts or control the transistor count to avoid overheating, limiting the chip's computing capability. Software-centric architecture is efficient for the programmers to code but not for the execution; similarly, hardware-centric architecture will perform exceptionally well for a particular domain but may not be suitable for general-purpose computation; hence it requires a combination of hardware-centric and software-centric architecture in a way that the software can reconfigure the hardware design according to the requirement.

2. Existing Solution

Most architects agree that meaningful gains in cost-energy quality now must come via domain-specific technology [9]. The existing solution is a specialized ASIC dubbed the TPU, which has been implemented in data centres since 2015 to expedite the inference process for neural networks (NNs). TPU's deterministic execution architecture is better for our NN applications' 99th percentile reaction time requirement than the time-varying improvements available on CPUs and GPUs, which provide more assured latency to throughput performance. The absence of such characteristics explains why, even though the TPU has many MACs and a large amount of memory, it is relatively tiny and has low power. Training (or understanding) and interpretation (or projection) are the two stages of NN relating to development vs production. The programmer decides the number of layers and the NN shape, while the weights are determined empirically. Quantization reduces floating-point values to small integer integers—often as little as 8 bits—generally sufficient for inference. Eight-bit integer multipliers may use up to 6X less power and 6X less space than IEEE 754 16-bit floating-point multipliers, resulting in 13X energy savings and 38X area savings for integer addition.

VeNNus [10] provides an open-source infrastructure for accelerating Ai systems. Analysis of the current and accessible AI accelerators in-depth and comprehended their significant characteristics. The VeNNus processor was designed with its primary capabilities in mind. This article discusses the design of the RISC-V Instruction Architecture-based VeNNus processor (ISA). RISC-V ISA provides flexibility, cheaper costs, and excellent efficiency, among other advantages. To boost the speed and energy consumption of our Ai-based (AI) accelerator, 16 new vector instructions were introduced to the RISC-V instruction set architecture. The VeNNus processor incorporates redundancy Arithmetic-Logic Units (ALUs) and compression of training weight to 8-bit integers to boost speed and throughput. With the CPU leveraging vector instruction and quantization, we predict that deep neural networks will be accelerated effectively. Variable predicate logic processors (VPLP) [11] include artificial intelligence (AI), robots, computer-assisted medicine, and electronic security. AI computer machines accomplish progress as an accelerating unit.

In contrast to established architectures, the data path of this processor is comprised of universal gates whose logical styles-subsets of predicated logic-change dynamically based on the data type and implemented instructions. In this study, a processor's reconfigurable gates and significant components are suggested, developed, modelled, and

confirmed to use a Field-Programmable Gate Array (FPGA) chip and a CAD tool. Using testing programs, the implemented processor validated its reconfigurability on-the-fly.

Neural network hardware accelerators are analyzed and categorized [12] according to the optimization approaches employed in their setup. Generally, each optimization strategy has an increase in specific performance parameters. Accelerators for deep neural networks are built using FPGAs. GPUs provide parallel computational units and parallelize neural network calculations. GPUs are power-hungry, which restricts their use in embedded devices. FPGAs offer high efficiency and are field customizable. FPGAs are frequently used for design prototyping and validation. ASICs are purpose-built for specific applications with optimal terms of power consumption. ASIC has additional embedded device applications. Talib et al.[13] analyzed a variety of accelerators for computer vision applications using FPGA, GPU, TPU and ASIC platforms and highlighted the benefits of each hardware compared to other hardware. Camus et al. [14] examined the accuracy of scalable MAC modules from several accelerators and described its advantages in various situations. In addition to the MAC unit, several additional accelerator parameters must be examined. 100% MAC use is required for an efficient architecture. Du et al. [15] outline a self-aware neural net in which a system may anticipate and change network parameter dynamics depending on input data. The self-aware approaches may considerably enhance the performance and energy efficiency of the accelerators, but the accelerators must be adaptable. To adapt and save energy, a neural network with varied accuracy requirements at various levels requires a MAC with variable precision.

2.1 TPU Architecture

Instead of being tightly tied with a central processor, the tensor processor was constructed as a co-processor here on the PCIe I/O bus, allowing it to plug into multiple servers similar to a GPU. Additionally, to enable hardware architecture and testing, the web host provides TPU instructions to the TPU rather than the TPU obtaining them. Consequently, the TPU is conceptually akin to an FPU (precision floating-point unit) co-processor rather than a GPU. The host transmits TPU instructions into an instructions buffer using the PCIe Gen3 x16 interface. Typically, the various blocks are connected using 256-byte-wide routes. The TPU's heart is the Matrix Multiple Unit. It includes 256x256 MACs capable of executing eight-bit multiplication and addition operations on signed or unsigned values.

The 16-bit products are collected in the matrix unit's four MiB of 32-bit Accumulators. 4096 256-element, 32-bit accumulators are contained in the 4 MiB. Each clock cycle, the matrix unit creates a 256-element partial sum. When 8-bit weights and 16-bit signals are used in conjunction, a Matrix is Essentially defined at half speed and a 1⁄4 speed if both are 16 bits. It can read and write 256 values per clock pulse and perform matrix multiplying or convolution. The matrix unit maintains a single 64 KiB tile with three weights and an extra one for the double buffering. This unit is meant to be used with dense matrices. For time-to-deployment issues, sparse architecture support was dropped. The matrix unit's weights are staged using an on-chip Weight FIFO that reads from an external 8 GiB DRAM termed Weight Storage. The FIFO for consequences is four tiles deep. The 24 MiB on-chip Unified Buffer holds the intermediate results, which may be utilized as inputs to the Matrix Unit. A customizable DMA controller is used to transmit data between the CPU Host and the Unified Buffer.

3. Proposed Work and Methodology

A high-speed block-level matrix multiplication unit is designed in the proposed design for accelerating the performance of convolution neural networks. This is based on the TPU architecture. Details about the proposed high-speed matrix multiplication unit are explained in section 3.1. To exploit the benefit of the parallel FPGA architecture proposed, a custom activation function is used. Section 3.2 describes in detail this custom activation function. A method of tree-based collective communication is used to optimize for accelerated performance. Details of the collaborative communication implemented are provided in section 3.3.

3.1 High-Speed Matrix Multiplication unit

Typically, matrix multiplication is a step-by-step multiply and adds process where each row and column of a matrix are multiplied and added to gather to get the multiplied output. In this paper, the idea is to design a custom matrix multiplication unit that bears all the values parallel in one clock cycle and performs the addition operation of the value in the next cycle in parallel.

The systolic array is a collection of processing elements that operate pipelined to calculate and transmit data through the system. A systolic array comprises many similar simple processors or processing elements (PEs) placed in a well-organized form, including a linear or two-dimensional array. Each processing element is interconnected with the others and has a finite amount of private storage. In our case, the processing element is composed of fused multiply and add units. For an ordinary 3X3 matrix multiplication, nine processing elements are arranged in a 2- dimensional array such that each processing unit corresponds to a component in the output array. Each partial output is stored in the processing element with a high-speed cache, reducing the primary memory access time. A systolic array drastically increases computation speed by reducing the memory access time.

3.2 Custom Activation Function in FPGA

In a deep net, the activation function is responsible for converting the node's cumulatively weighted input to node and output activation for that input. The fundamental concept is implementing a rectified linear (ReLU) activation function in hardware (FPGA) and using parallelism wherever feasible. Since the network that employs it is faster to train and often delivers more excellent performance, it has become the standard activation element for many types of neural networks.

3.3 Collective Communication in FPGA

As explained in the last section, FPGA has multiple partitions capable of performing multiplicative and additive operations. From each PE, there is a hierarchical communication with other PE. This communication is initialized during a neural network implementation and then fixed for the rest of the operation. A partially reconfigurable controller is used to establish the PE and the communications. A detailed communication diagram for collective operations like reduce, gather, and scatter is shown in the figure. Reduce is used for the neural network's weight calculation at each iteration. Some PE elements will use spread for sharing their weights calculated and gather will be used by all the nodes at the end of each iteration to update themselves with the weight value computed. In Figure 2, the communication is explained. Algorithm 1 is the proposed algorithm to configure the FPGA and execute the CNN considered. In Figure 2, each cell represents a processing element.

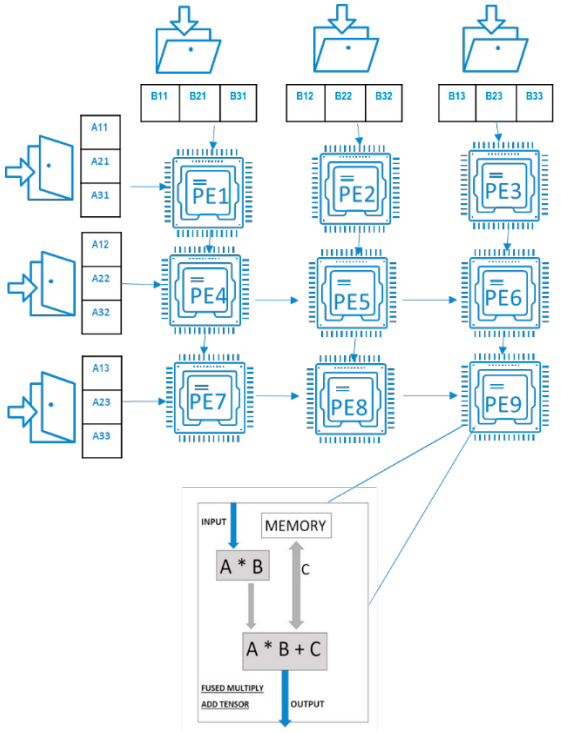

Fig. 1. The systolic array of processing elements connected for a sample 3*3 matrix multiplication in FPGA

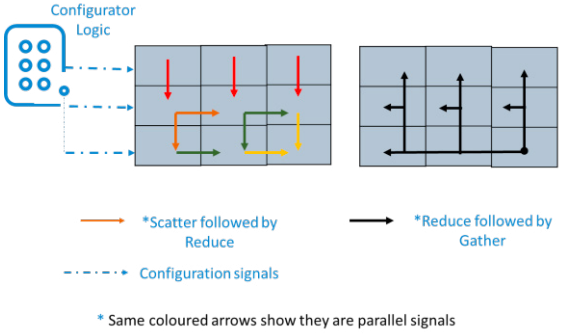

Fig 2. Communication Hierarchy for a sample of 9 PEs in FPGA

Algorithm 1

Accelerating DNN in FPGA through proposed Soft-Core Processor

Initial Condition:

- CPU initializes the Neural Network execution process
- CPU communicates with the Configurator logic on the FPGA through PCI express
- Configurator logic configures as many Processing Elements (PE) as needed and will establish a communication between them

Input:

A, B: 2-dimensional matrices with (i*j) index where i and j can vary from 1 to 'n'

Result: O, a matrix with n,n dimension; Product of A and B

Configuration Logic:

```
{
Configurator in FPGA is implemented with LUTs and MUXes.
Also connected to the sequential storage for acceleration
I= [(i,j)← 0]
AC = [(i,j)← 0]
O = [(i,j)← 0]
ProcessingElement(A, B, 0, I, A, C, O)
)
```

Processing Element Computation (A, B, k, I, AC, O):

```
{
If (k=3n-1) then
        Return 0;
Else for partition k
        {
        I'=[(n,n) → (i,0), then, B(k-j,j):I(i-1,j)]
        AC'=[(i,j) → (j,0), then AC(i,j-1): A(i,j)*I'(I,j)
        O'=O[i,(I,k-n-i)→AC(i,n-1)
        PE(A,B, k+1,I',AC',0)
        }
Communicate with Partition k' where k' value vary from 1 to 3n-1
}
```

Communication Logic(k)

```
{
If (k==0||1||2, k'= {k+1})
If (k==3|4|5…n-3, k'= {k+1, k+2}
If (k==n-2|n-1|n, k'= {k+1}
Communication from partition k to all partitions in k'
}
```

4. Implementation

Vivado, a high-level synthesis, is used to develop a high-speed tensor multiplication IP. Vivado enables rapid design implementation by directly targeting C and C++ specifications to Xilinx devices, even without requiring manually generated RTL. High-level synthesis establishes a link between the hardware and software worlds. Operate at a higher degree of abstraction in the hardware while still producing high-performance technology. While in the software sector, FPGA IP blocks may be employed to expedite the computationally expensive aspects of their algorithms. Sequencing, bonding, and control logic are all components of high-level synthesis. Scheduling decides which activities occur throughout each clock cycle depending on the cycle's frequency. Binding specifies the hardware resource that will be utilized to execute each scheduled operation. The control logic is used to construct a finite state machine (FSM) to perform the RTL operations sequentially. High-level synthesis converts top-level function parameters to RTL I/O ports. All input and output actions in the 'C' language are accomplished through formal function parameters. Whereas with an RTL design, these identical input and output activities must be conducted through a port inside the design and often use a unique I/O protocol. Use of array variables as the function argument's input and output for our Matrix multiplication IP design. Array variables are by default implemented as read-write RAMs. However, if the data in the array is read or created in a continuous series of actions, a more effective communications technique is streaming data, which utilizes FIFOs rather than RAMs.

As a consequence, employing a single AXI4 streaming interface to input both arrays and another AXI4 streaming interface to output the multiplied result of the two input arrays. The AXI4 stream is linked to the ZYNQ processing

system IP through one of the ZYNQ architecture's four available high-performance ports. The ZYNQ processing system IP encapsulates the Zynq7000 architecture's software interface. The processing system IP wrapper links the ZYNQ processing and the programmable logic, which is a wrapper for the Field-Programmable Gate Array (FPGA).

When building a Systolic array design for tensor multiplication, the IP assignment and scheduling are accomplished using a loop with the PIPELINE pragma for Hls design. By permitting parallel execution of operations, the PIPELINE pragma decreases the start time for a function or loop. Pipelining a loop enables concurrent execution of the loop's actions. The PIPELINE pragma's default initiation interval is 1, which processes one incoming information to the processor five elements (PEs) per clock cycle. Internally, the PIPELINE pragma rolls or unrolls the looping or recursive function. When loops are moved, the logic with one loop generation is generated via synthesis, and the matching RTL design is executed sequentially for every iteration of the circle. The processing element of a systolic array may unroll loops through optimization directives, allowing for parallel execution of all iterations. An overview of the implementation is shown in Figure 3.

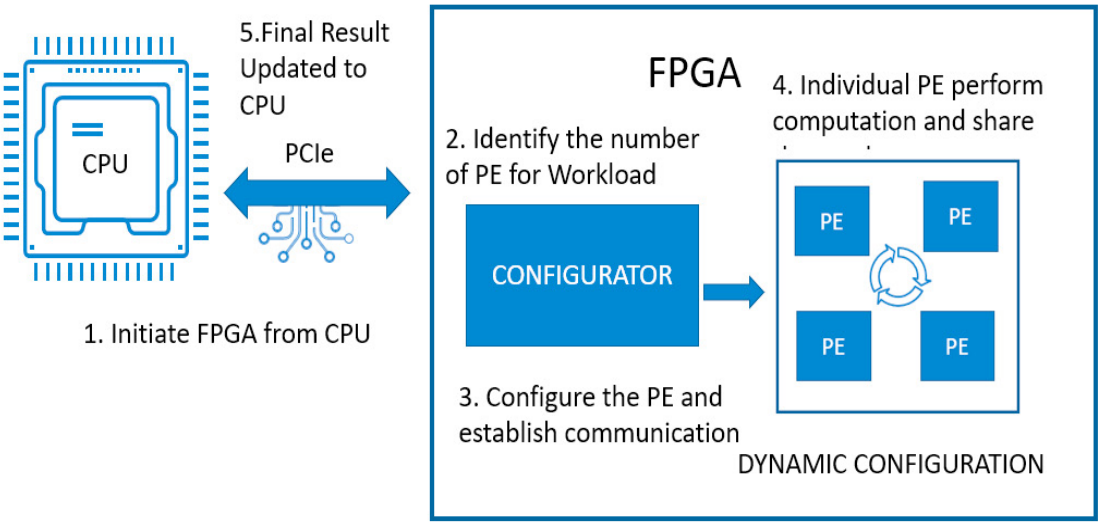

Fig. 3. Proposed Model

4.1 Systolic Array Multiplication Algorithm

The algorithm for matrix multiplication using a systolic array is given in Algorithm 1. The input n represents the total number of processing elements. The condition k=3n−1 gives you the number of times the two matrices will intersect with each other. For example, consider a 4×4 matrix. Then there will be 11 intersection, multiplication, and accumulation operations in the processing unit of a systolic array. [(i,j)↦0]: means an assignment of value zero to the element (i,j)(i,j) of the matrix; it is used to initialize the temporary matrices. I′ in line 11 is a provisional matrix that collects matrix B elements at the intersection of the two matrices (elements ready for performing operations). Thus, the relevant input of B is denoted by I′ . AC′ in line 12 is an accumulator. Each element A(i,j) is multiplied by each component of I′ (i,j). Recall that I′ (i,j) is obtained from B. Multiplying B and A differently. O′ in line 13 is the output matrix. It is calculated by moving the accumulated values of AC′ to their correct positions respective to their processing elements. ii. Zynq System After creating the custom IP, the designed IP core must be imported into the vivado design suite under the new block design. Next ZYNQ processing system is added, and a connection automation procedure is followed to connect the board to the vivado design suite. Now there is a need to configure the interfaces of the ZYNQ processing system for interfacing our custom tensor multiplication and activation function IP created in vivado HLS. Since our custom IP uses steaming interfaces, we use DMA connected to the high-performance port of the ZYNQ processing system. Direct Memory Access (DMA) transfers the block of data between the system's memory and peripheral devices without the processor's active participation. In ZYNQ architecture shown in Figure 4, DMA

transfers data from the DDR3 memory to the required IP core in the programmable logic via the high-performance port.

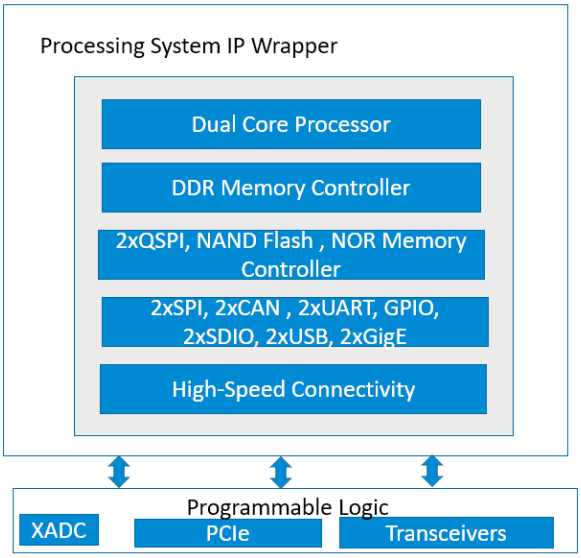

Fig. 4. ZYNQ processing system

5. Results and Evaluation

To simulate the working of the matrix multiplication IP core, use test bench code written on VHDL with a custom 3*3 matrix. The result is observed on the timing diagram of the simulation. From the timing diagram, it can be seen that every value of the output c is a cumulative addition of values from the starting of the clock signal. The Timing diagram and comparing the results of traditional matrix multiplication showed that a 3*3 matrix requires 27 clock cycles to get the output. In contrast, using the custom IP, the clock cycle has been reduced to 8. Since the Pynq FPGA board uses a clock of 128 MHz, it takes 7.76 ns to complete one clock cycle; the total time required to complete a 3*3 matrix multiplication is about 7.76 * 8 = 62.08 ns.

Table 1. Comparison of Execution time with and without hardware

Dimension of Input Kernel	Without Hardware Acceleration		With Hardware Acceleration	
	Number of Clock Cycles	Time(ns)	Number of Clock Cycles	Time(ns)
3x3	27	209 ns	8	62 ns
4x4	64	496ns	11	85ns
8x8	512	3973ns	23	178ns
16x16	4096	31784 ns	47	364 ns

In **[16]**, the geospatial information system's floating-point-intensive kernel was done with an evaluation of both kernel efficiency as well as the floating-point error rate of the FPGA designs that were developed. In this single precision and double precision floating point unit in FPGA was implemented, the proposed model is compared with this work and the performance is shown in Figure 5.

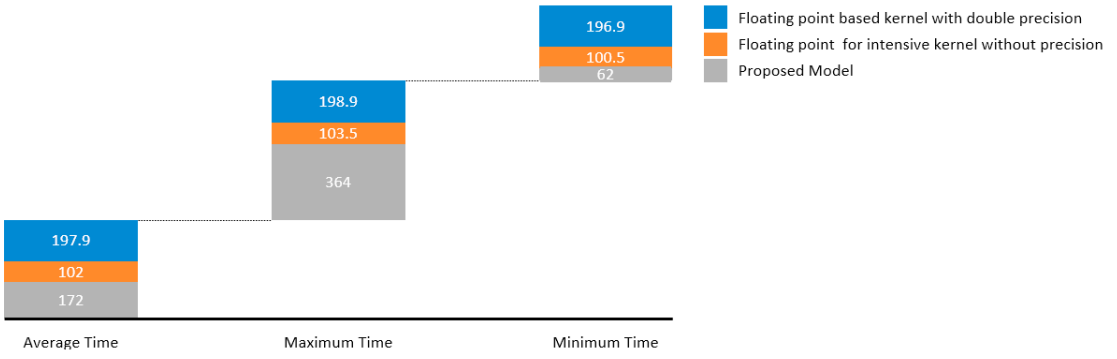

Fig. 5. Comparison with the existing method

6. Conclusion

Separating the high computational process from the main CPU by creating custom computational hardware for Artificial neural networks and deep learning. This also results in the CPU's performance gain with reduced power requirements. Due to the increased use of machine learning in our cell phones and laptops, there is a need for a separate processing system since these devices are often underpowered. The paper describes the advantages of a simple cascaded multiplication unit in the co-processor. The results show significant improvement in the speed of the floating-point matrix multiplication operation. Further, the co-processor can be fitted with hardware-implemented neural network layers for simple tasks which can be used in a much deeper network. For example, the ideology can be used to create the hardware-implemented neural network with few layers for feature extraction, which can be used along with transfer learning and can be used in any computer vision application.

Acknowledgements

We want to express our gratitude to Mr Krishna. V, a student of Vellore Institute of Technology, Chennai, India, for his exceptional support in completing this work.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems*, 2012, vol. 25. Accessed: Feb. 21, 2022. [Online]. Available: https://papers.nips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html

[2] H. A. Pierson and M. S. Gashler, "Deep learning in robotics: a review of recent research," *Advanced Robotics*, vol. 31, no. 16, pp. 821–835, Aug. 2017, doi: 10.1080/01691864.2017.1365009.

[3] D. S. Berman, A. L. Buczak, J. S. Chavis, and C. L. Corbett, "A Survey of Deep Learning Methods for Cyber Security," *Information*, vol. 10, no. 4, Art. no. 4, Apr. 2019, doi: 10.3390/info10040122.

[4] M. Havaei *et al.*, "Brain Tumor Segmentation with Deep Neural Networks," *Medical Image Analysis*, vol. 35, pp. 18–31, Jan. 2017, doi: 10.1016/j.media.2016.05.004.

[5] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "DeepDriving: Learning Affordance for Direct Perception in Autonomous Driving," in *2015 IEEE International Conference on Computer Vision (ICCV)*, Dec. 2015, pp. 2722–2730. doi: 10.1109/ICCV.2015.312.

[6] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, Dec. 2017, doi: 10.1109/JPROC.2017.2761740.

[7] D. Patterson, "50 Years of computer architecture: From the mainframe CPU to the domain-specific tpu and the open RISC-V instruction set," in *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*, Feb. 2018, pp. 27–31. doi: 10.1109/ISSCC.2018.8310168.

[8] J. Dean, D. Patterson, and C. Young, "A New Golden Age in Computer Architecture: Empowering the Machine-Learning Revolution," *IEEE Micro*, vol. 38, no. 2, pp. 21–29, Mar. 2018, doi: 10.1109/MM.2018.112130030.

[9] N. P. Jouppi *et al.*, "In-Datacenter Performance Analysis of a Tensor Processing UnitTM," p. 17.

[10] S. Harini, A. Ravikumar, and D. Garg, "VeNNus: An Artificial Intelligence Accelerator Based on RISC-V Architecture," in *Proceedings of International Conference on Computational Intelligence and Data Engineering*, Singapore, 2021, pp. 287–300. doi: 10.1007/978-981-15-8767-2_25.

[11] A. N. Kostadinov and G. A. Kouzaev, "A Novel Processor for Artificial Intelligence Acceleration," *WSEAS TRANSACTIONS ON CIRCUITS AND SYSTEMS*, vol. 21, pp. 125–141, Jul. 2022, doi: 10.37394/23201.2022.21.14.

[12] R. Machupalli, M. Hossain, and M. Mandal, "Review of ASIC accelerators for deep neural network," *Microprocessors and Microsystems*, vol. 89, p. 104441, Mar. 2022, doi: 10.1016/j.micpro.2022.104441.

[13] M. A. Talib, S. Majzoub, Q. Nasir, and D. Jamal, "A systematic literature review on hardware implementation of artificial intelligence algorithms," *J Supercomput*, vol. 77, no. 2, pp. 1897–1938, Feb. 2021, doi: 10.1007/s11227-020-03325-8.

[14] V. Camus, L. Mei, C. Enz, and M. Verhelst, "Review and Benchmarking of Precision-Scalable Multiply-Accumulate Unit Architectures for Embedded Neural-Network Processing," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 4, pp. 697–711, Dec. 2019, doi: 10.1109/JETCAS.2019.2950386.

[15] B. Z. Du, Q. Guo, Y. Zhao, T. Zhi, Y. Chen, and Z. Xu, "Self-Aware Neural Network Systems: A Survey and New Perspective," *Proceedings of the IEEE*, vol. 108, no. 7, pp. 1047–1067, Jul. 2020, doi: 10.1109/JPROC.2020.2977722.

[16] Z. Jin, H. Finkel, K. Yoshii, and F. Cappello, "Evaluation of a Floating-Point Intensive Kernel on FPGA," in *Euro-Par 2017: Parallel Processing Workshops*, Cham, 2018, pp. 664–675. doi: 10.1007/978-3-319-75178-8_53.