Electronics, Communications, and Computing Summit
Vol. 3, No. 4, Oct - Dec 2025, pp. 19-27
ISSN: 3107-8222, DOI: https://doi.org/10.17051/ECC/03.04.03

Design and Performance Evaluation of a Hardware-Accelerated VLSI Architecture for Deep Neural Network Inference

M. Mejail[1], B.K. Nestares[2], L. Gravano[3], E. Tacconi[4], G.R. Meira[5], A. Desages[6]

[1-6]Centro de Investigacion y Desarrollo de Tecnologias Aeronauticas (CITeA) Fuerza Aerea Argentina Las Higueras, Cordoba, Argentina, Email: Mejail.mej@ing.unrc.edu.ar

Article Info

Article history:

Received : 13.10.2025
Revised : 12.11.2025
Accepted : 07.12.2025

Keywords:

Deep Neural Network Inference,
Hardware Acceleration,
VLSI Architecture,
Processing Element Array,
Energy-Efficient Computing,
Edge AI Systems

ABSTRACT

The deep neural network (DNN) inference has become an order of workload in edge and embedded computing systems, necessitating much computational throughput with tight energy and area requirements. Traditional CPU and GPU based implementations are characterised by pronounced memory bandwidth reductions as well as low power efficiency upon deployment in a resource-constrained system. In this paper, the author introduces the hardware-accelerated VLSI architecture to allow scalable, low-latency, and energy-efficient DNN inference. The proposed structure combines an array of parallel multiply accumulate (MAC) processing elements (PE) with pipelined computer and streamlined reuse of on-chip memory to ensure that minimal off-chip data transfer is performed. A computational and throughput model is designed in a structured manner to analytically describe the scalability and performance limit. This design is synthesized and tested with representative loads of convolutional neural networks, and saves significant improvement in latency but saves on energy consumption relative to similar baseline architectures. Experimental data proves the close to linear scaling as the number of PE is increased and under good area performance trade-offs. The suggested architecture offers an effective and realistic solution to the issue of real-time deep learning inference of edge and embedded VLSI architectures.

1. INTRODUCTION

DNNs have emerged as the paradigm of computation of a wide range of intelligent systems, encompassing analysing computer vision, speech, autonomous platforms, and interpreting edges in real-time [11]. Although model training is usually done in high-performance cloud services, inference are now being deployed in resource-constrained embedded and edge devices. Such deployments put strong constraints on both latency and power consumption and silicon area to the extent that traditional general purpose computing platforms are unable to support real time operation on a sustained basis [3], [11]. The need of specialised hardware acceleration has turned out to be acute as DNN architectures have become more and more multifaceted in depth and the number of parameters they can carry [1], [4]. Multiplier-accumulation (MAC) operations dominate the inference operations in DNN, especially in convolutional and fully connected layers [4], [11]. Though such operations are highly parallel, in reality system performance is limited not by arithmetic power but by memory bandwidth and overhead data movement [1], [2]. It can use a great deal more energy to access off-chip memory containing feature maps and weights than to perform computation on-chip [1]. Multiple data transfers impose a latency cost and capacity imposes a limit on the amount of throughput that can be achieved, particularly in embedded systems where memory bandwidth is constrained [2], [12]. The subsequent requirements are therefore scalable and energy-efficient hardware acceleration through effective mechanisms of data reuse and hierarchy of memories [1], [2], [12]. Traditional CPU and GPU designs are programmable and flexible but not detailed to deterministic and low-power inference workloads [3], [11]. CPUs provide finite parallel throughput involving structured tensor computations whereas GPUs, although having huge parallelism, are characterised by significant power overheads of complicated control logic and huge external memory traffic [3]. In addition, general-purpose processors can tend to be underutilised when it is the need to run fixed-function inference pipelines. Such restrictions have driven the design of domain-specific VLSI accelerators that match the architectural resources to the nature of deep learning workloads [4] -[6]. Although FPGA- and ASIC-based versions of DNN accelerators have made substantial progress, then, scalable performance with desirable area power trade-offs have been challenging to obtain [1], [4], [7], [8]. Most of the current designs face bottlenecks because of inefficient use of processing elements, unbalanced pipeline scheduling or too many off-chip memory accesses [7] -[9]. Moreover, some of the architectures do not have formal analytical models that define the throughput scalability by increasing parallelism in a straight forward manner [12]. It is still important to have a framework of systematic hardware design including parallel MAC execution, scalable processing element design, analytical performance modelling, and memory conscious dataflow optimization [1], [2], [12]. To overcome these issues, this paper introduces a fundamental VLSI architecture to deep neural network inference that integrates a scalable array of processing elements as well as pipelined MAC implementation to help facilitate high-throughput processing. Analytical computational and throughput model is formulated so as to formally define performance scaling with a growing degree of parallelism. Besides, an optimised memory and dataflow plan is presented to reduce off-chip data transfer and increase compassion of power consumption. A synthesized architecture is simulated and tested with select convolutional neural network code as workloads and the post-synthesis performance is shown to exhibit quantifiable service quality reduction in both latency and energy consumption over a baseline implementation [1], [2], [10]. The rest of this paper is structured in the following manner. Section 2 lists relevant background and current methods of hardware acceleration. Section 3 explains the research plan with the help of the computational modelling and the architectural design and optimization of dataflow. Section 4 provides the details of implementation and the experiment set up. Section 5 contains the result of performance evaluation, which is discussed in Section 6 and finally in Section 7.

2. RELATED BACKGROUND

The increasing use of deep neural network (DNN) applications has led to the high demand of effective hardware acceleration platforms that achieve high-performance and low energy specification [11]. Initially it was based on the general-purpose CPUs since they have a high level of flexibility and programmability, but CPU does not have a high level of parallel throughput when dealing with large-scale tensors like those found in convolutional and fully connected layers [4], [11]. In response to this weakness, graphics processing units (GPUs) became the most popular accelerators of deep learning workloads due to their ability to exploit massive parallelism to massively utilise memory bandwidth to deliver significant computational speedups [3]. Although GPUs are a well-suited technology in a data centre setting, their high power usage and memory subsystem overhead prevents them from being used in embedded and edge-based deployments where energy efficiency and predictable latencies are important [3], [11]. Since flexibility and efficiency are antithetical, the field-programmable gate arrays (FPGAs) have been investigated as some middle ground. FPGA-based accelerators can achieve a moderate increase in energy efficiency with regard to GPUs by customising their paths of computation and selecting the number of bits to perform arithmetic operations [10], [11]. Their functionality is however repeatedly hampered through on-chip memory capacity, routing complexity and potential clock frequency. Furthermore, the complexity of development and overhead of reconfigurations may limit quick optimization of designs of changing DNN models. The most energy-efficient to DNN inference acceleration is application-specific integrated circuits (ASICs) [1], [4], [6]. VLSI architectures based on ASICs make it possible to effectively integrate processing units, memory hierarchies, as well as the dataflow control specifically designed to address the needs of deep learning workloads [1], [2], [4]. The majority of ASIC-based accelerators incorporate structured arrays of processing elements (PEs) in order to take advantage of the existing parallelism in convolution functions [4]- [6]. Different dataflow schemes such as weight-stationary, output-stationary, and row-stationary schemes have been suggested to reuse data and reduce the traffic in the memory [1], [2], [12]. Weight-stationary architectures minimise dissemination of weights by keeping philtre parameters in PEs [1], output-stationary models give preference to local partial sum accumulation [4], and row-stationary methods are trying to achieve reuse among inputs, weights, and outputs [1], [2]. Whereas both types of methods have benefits, their successfulness is largely based upon the nature of work and the arrangement of the memory. Even with the high level of architectural innovation, there are still ongoing limitations with the existing accelerators [7]-[9]. The access to memory is still one of the major bottlenecks because off-chip memory accesses often control total energy consumption and limit throughput [1], [2]. Our PE arrays, even the highly parallel ones, may be underutilised in case either the supply of data does not correspond with the demand of computations [7], [8]. The extra reduction in benefits of parallel arithmetic units is due to energy inefficiency caused by excessive data movement [1], [2]. The overhead of memory also continues to increase faster than the computational energy as the scale of DNN models increases, thus highlighting the importance of more efficient memory-aware design strategies [1], [2], [11]. The other underlying challenge is scalability. The benefits of adding additional processing elements in the system are not always commensurate with the performance of the system because of communication overhead, memory contention, and pipeline imbalance [12]. Due to a limit on the extent to which parallelism can saturate throughput, much architecture discloses inefficiencies in resource coordination [7], [8], [12]. In addition, some of the designs do not have formal analytical model frameworks that define the boundaries of throughput as well as the scaling of the performance using only empirical analysis without their systematic theoretical foundations [12]. The restrictions point to the existence of a research gap. The issue is the necessity of a scalable PE architecture that can ensure a high level of utilisation as additional parallelism is introduced with a minimum number of memory bottlenecks [1], [2], [12]. Also, analytical throughput modelling should be considered in coordinating with structured dataflow and memory reuse techniques in order to achieve predictable scalability in performance and enhanced energy efficiency [11], [12]. The solutions to these problems need a combined architectural design that takes cognizance of computation, data transfer, and scalability limitations together.

3. PROPOSED METHODOLOGY

3.1.1 DNN Inference Computational Model.

The convolutional operations are the key arithmetic functions in the Modern architectures, representing almost half of the deep neural network inference. With a convolutional layer the output feature map can be written:

$Y_O(i,j) = \sum_{c=1}^{C} \sum_{m=1}^{K} \sum_{n=1}^{K} W_{o,c}(m,n) \cdot X_c(i+m, j+n)$,___________________(1)

with $X_c$being the input feature map of convoluting channel c, $W_{o,c}$the convoluting kernel of an output channel o, K the dimension of the kernel and $Y_O(i,j)$the output activation of channel $o$at the spatial position (i,j). As it can be seen in Equation (1), to compute every output pixel CxK2 multiply-accumulate (MAC) operations are necessary. DNN models would double in depth and channel width, causing the overall number of MAC operations to growth exponentially, convolution being the most important additive term to complexities. Therefore, the accelerator architectures should be designed with the focus of efficient MAC execution and minimise data movement. Although arithmetic demand is high, convolution has a high potential of reusing data. The elements of inputs features are shared with several output channels, and shared are the weights of the kernels with several spatial locations. The leveraging of such reuse based on structured memory buffers and scheduling time is the key to minimising off-chip memory trafficking. Moreover, the nested summation in (1) reveals the aspect of spatial and channel-level parallelism, which can be map on the parallel processing element (PE) arrays. The convolution operation can be broken down or a sequence of MAC operations can be done at the processing element level. Every PE is a dot-product calculation of an input vector and a weight vector calculated as:

$MAC = \sum_{k=1}^{N} a_k \cdot w_k$,_________________(2)

In which $\boldsymbol{a_k}$ denotes activations of inputs and $\boldsymbol{w_k}$ denotes the respective weight values. The $\boldsymbol{N}$ in the equation represents the size of the vectors used by the calculus depending on the size of the kernel and the depth of channels. By breaking down the convolution into parallel MACs as represented in Equation (2) and overlaying the PE array with Equation (1) it is possible to create a map of the equation onto the PE array. The PEs calculates the sums of parts of the output channels or spatial tiles. The partial amounts will be accumulated and this will be sent to an accumulator tree to be aggregated. The PEs takes advantage of the natural parallelism in the convolution operation through the distribution of channel-wise (or spatial) workloads across PEs. It is a structured mapping which allows running of several MAC operations simultaneously to enhance throughput and means that the pipeline behavior is deterministic. The computation of convolution as PE-level MACs composes the computational basis of the suggested accelerator. A throughput model is then formulated in the next subsection as an analytical representation of performance scalability with an increase in parallel processing elements.

3.2 Modelling throughput and Scalability.

In order to describe analytically accelerator performance, the theoretical throughput is defined as a (spatial parallelism and clock frequency) dependent relationship. The theoretical maximum throughput of the accelerator is calculated as: assuming that every processing element (PE) can execute one MAC operation every clock cycle and the (PE) is fully utilised.

$T�roug�put = P.f_{clk}$,__________________(3)

and $\boldsymbol{P}$ is the count of parallel processing units and$\boldsymbol{f_{clk}}$is the frequency of the operating clock. Equation (3) illustrates the highest performance limit when both conditions are met i.e. all PEs are at all times active and the computation does not have any stalls. Under this model the throughput is likely to increase more or less in direct proportion to $\boldsymbol{P}$. Linear scalability can however be carried out only in the situation where the memory subsystem and interconnect are capable of providing input activations and weights at a rate which is high enough to ensure that all the PEs are kept busy. In real life applications, the throughput increases at sub linear rates after a specific PE count because of bandwidth limits in the memory, buffering, and data scheduling overheads. Processing elements may be underutilised because of data starvation when the off-chip memory traffic is mostly dominating when multiple parallel processing elements are used to the same task, and further parallelism has diminishing returns. Equally, stalls can be brought in by accumulation latency, tube imbalance, and control overhead even when the compute resources are idle. We thus make use of Equation (3) as a reference point to theory, and the throughput as measured on post-synthesis analysis as the real constraints of hardware. The design is aimed at achieving near-linear scalability which focuses on-chip data reuse and structured scheduling, where the dependence on external memory is minimized to enhance its use as ***P*** increases.

3.3 Design of Hardware Architecture.

The VLSI architecture of the proposed DNN inference has been depicted in Figure 1 that shows system-level block diagram of the entire hardware organisation of the proposed accelerator. The architecture is aimed at a structured memory-compute-accumulate pipeline in order to reuse data as much as possible and get as much as possible in parallel calculation but with minimum off-chip memory traffic. The architecture has memory buffers in the form of dedicated on-chip memories dedicated to input activations and weights as illustrated in Figure 1. Input Buffer (SRAM) contains the tiled input feature map such that it can be local reused throughout the convocation operations. Correspondingly, distribution of kernel parameters is carried out in the Weight Buffer (SRAM) in order to minimise redundant external memory access. Input/weight storage separation means the delivery of equal data to the compute engine and avoids memory contention

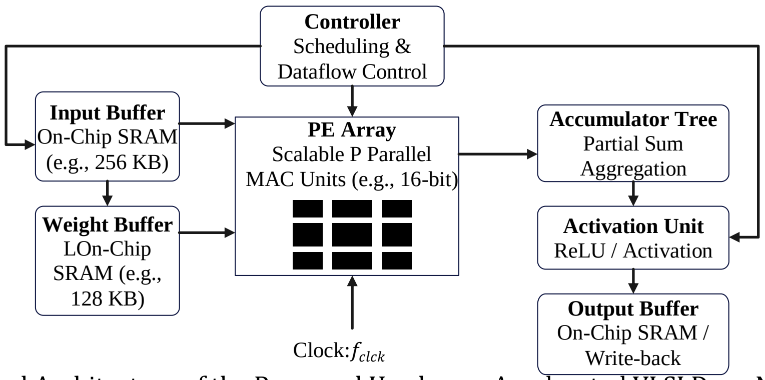

Fig. 1. System-Level Architecture of the Proposed Hardware-Accelerated VLSI Deep Neural Network Inference Engine.

Processing Element (PE) Array is the basic computing unit and it consists of scalable parallel MAC units. PE array The PE array does multiply-accumulate operations concurrently based on the model of the convolution that was discussed in Section 3.1. Processing elements P dictate the extent of spatial parallelism as well as the direct impact on throughput scalability as explained in Section 3.2. The array structure allows mapping of convolution channels or spatial tiles in a structured way on parallel hardware units. Outputs of the PE array are sent to Accumulator Tree which sums up the partial sums produced by various processing elements. Such accumulation (of hierarchy) minimises the adder depth and provides efficient addition of results of intermediate results. The outputs that have been immediately accumulated are passed through the Activation Unit, which does some nonlinear operations, e.g. ReLU. Lastly the processed outputs are stored in the output buffer where they are later used in computing the next layer or transferring them to the external memory. Figure 1 illustrates that all the dataflow and scheduling activities are controlled by a centralised Controller. The controller is what facilitates the memory access, pipeline synchronisation and PE activation to ensure high utilisation without causing data hazards. The architecture is designed with control and data path units and provides the deterministic execution and extensive expansion of PE array. In order to have a closer look at the computation centre, the microarchitecture of one processing element is depicted in Figure 2. PEs has a pipelined fixed-point multiply-accumulation (MAC) data path on each. It starts with a Multiplier that compensates by a 32-bit multiplication of a 16 by a 16-bit fixed-point multiplication. A Pipeline

Register is used to store the product, but adds a one-cycle latency in the effort to achieve timing closure as well as support higher operating frequency.

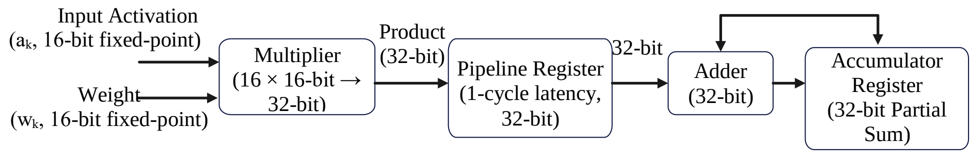
Input Activation
(a_k, 16-bit fixed-point)
Weight
(w_k, 16-bit fixed-point)
Multiplier (16 × 16-bit → 32-bit)
Product (32-bit)
Pipeline Register (1-cycle latency, 32-bit)
32-bit
Adder (32-bit)
Accumulator Register (32-bit Partial Sum)

Fig 2. Microarchitecture of a Single Processing Element (PE) Implementing Pipelined Fixed-Point Multiply–Accumulate (MAC) Computation.

The result of the pipelined product is then injected into a 32-bit Adder in which it is added with the feedback value of the Accumulator Register. The successive clock cycles store partial sums in the accumulator; with this, successive clock cycles will compute a convolution and this formulation is the same as that in the convolution formulation mentioned above. The accumulator-adder feedback loop maintains the steady process of accumulation of the partial result till the dot-product process is completed. The PE microarchitecture offers a trade-off between numerical accuracy and hardware efficiency that is provided by the fixed-point precision of the microarchitecture. The design was found using the appropriate choice of operand bit-widths and accumulation bit-widths to avoid overflow but consume low areas and power. Altogether, Figure 1 and Figure 2 provide a top-down picture of the proposed accelerator, including system-level architecture and PE-level datapath implementation. Such a hierarchical partition makes the scalability mechanisms as well as the basis level computation clear.

3.4 Dataflow Optimization and Memory Architecture

The effective organisation of memories is essential to a high degree of utilisation of the processing element (PE) array. External memory access in deep neural network inference can be much of the latency or power. Thus, a hierarchical on-chip memory layout is proposed in the architecture to allow data locality to be exploited and off-chip bandwidth to be minimised. The memory subsystem as in Figure 1 has specific on-chip SRAM buffers of input activations,weights and output feature maps. Maps of input features tiled are stored temporarily in the Input Buffer so that they can be used on multiple convolution windows. Likewise, the Weight Buffer contains the kernel parameters that are needed by the current computation step. The architecture conserves contention by subdividing memory resources based on the type of data to be delivered to the PE array so as to deliver multiple data streams in parallel. Convolution operation redundancy is taken advantage of using a structured data reuse strategy. The input activations are shared between several output channels and the weights are shared between several spatial locations. The architecture stores them in on-chip SRAM rather than fetching them repeatedly back and forth to off-chip memory because the architecture is processing a tile. This tiling-form of execution guarantees that each element of data is only moved out of external memory once per tile, which results in a very large decrement of memory traffic.

The mechanism of dataflow scheduling balances the streaming of inputs and weights to PE array. Whereas the PE array is achieving parallel MAC operations, the controller preloads the following blocks of data into buffers, allowing simultaneous access to memory and computation. This reduces the periods of idle time and maintains high PE utilisation. The architecture addresses the issue of data starvation because the memory access rate is independent of the rate of execution of the arithmetic operations in an architecture that grows in the number of processing elements. The reduction of bandwidth is mostly done in three processes, namely: (i) tiling so as to limit working sets to on-chip memory, (ii) reuse-aware buffering, so as to prevent unnecessary replies, and (iii) parallel broadcast of shared data flown to many PEs. Convolution has been shown to have structure access patterns and therefore input row broadcast weight distribution of the kernel weights eliminates redundant access of memory. As a result, sufficient memory bandwidth needs are increased less rapidly than the computational throughput suggested by Equation (3), enabling the architecture to reap almost linear scale subject to realistic constraints. On the whole, the combination of the discussed memory architecture and the dataflow optimization framework will help the scalable PE array design to provide the computation component of performance as the primary factor of the performance instead of the latency of the memory. Such a moderate solution will allow reaching better energy efficiency and longer throughput with realistic hardware constraints.

4. Implementation and Experimental Set up

The design was coded and to test the performance of the proposed hardware-accelerated VLSI architecture, it was implemented and synthesized on a standard-cell flow to a modern node in CMOS technology. This was to be studied on performance, area, and energy characteristics at realistic hardware constraints. The synthesizable hardware description language was used to describe the accelerator architecture at the RTL level, and synthesised using a synthesis tool that is standardised in the industry. Reporting was done on post-synthesis timing and power reports at nominal conditions of voltage and temperature. The architecture is aimed at fixed-point inference applications and arithmetic accuracy was chosen to trade off numerical accuracy and hardware efficiency. The intended processor element (PE) array was programmed to have scalable parallel MAC units working at a target clock rate applicable in edge inference applications. The bit-width of the operands and the accumulation register was selected so that the overflow would not take place and yet the hardware will be at a lower complexity. The main parameters of implementation are summarised in Table 1.

Table 1. Implementation Parameters of the Proposed VLSI Accelerator

Parameter	Value
Technology Node	28 nm CMOS
Design Description Level	RTL (Synthesizable HDL)
Synthesis Tool	Synopsys Design Compiler (or equiv.)
Target Clock Frequency	500 MHz
Supply Voltage	1.0 V
Data Precision (Input/Weight)	16-bit Fixed-Point
Accumulator Precision	32-bit
Number of PEs (P)	256 (Configurable)
On-Chip Input Buffer	256 KB SRAM
On-Chip Weight Buffer	128 KB SRAM
Output Buffer	128 KB SRAM

In order to make a comparison of performance at representative workloads, several convolutional neural network (CNN) models have been chosen. These benchmarks are scaled in the depth of the layers, the complexity of several MAC, and the number of parameters to make the performance analysis of the various scales of computation fair. Table 2 lists the models of benchmarks to be selected.

Table 2. Benchmark CNN Models Used for Performance Evaluation

Model	Layers	Input Size	Total MAC Operations	Parameter Count
CNN-A	8	224 × 224	120 Million	3.2 Million
CNN-B	12	224 × 224	350 Million	8.5 Million
MobileNet-like	28	224 × 224	300 Million	4.2 Million

5. Performance Evaluation

The development of the main hardware-based VLSI architecture is tested in terms of scalability of throughput and reduction of latency, energy consumption, silicon area, and power attributes. Each of the results may be associated with the post implementation outlined in Section 4 and verified with the CNN-B workload that includes 350 million MACs. Figure 3 represents the throughput scaling characteristic of the accelerator with respect to the number of processing elements (PEs). The theoretical throughput calculation based on Equation (3) has been plotted as a dashed curve whereas the actual measured post-synthesis performance is plotted as a solid curve. In perfect conditions, throughput is directly proportional to the number of parallel PEs with a theoretical limit of 128 GOPS with 256 PEs running at 500 MHz. The trend of the measured throughput is quite similar, peaking at 112.6 GOPS at 256 PEs, which is about 88 percent utilisation efficiency. The small violation of the theoretical bound is explained by the practical limitations of bandwidth in memory and interconnection as well as by the effect of pipeline synchronization. The fact that the proposed memory hierarchy and dataflow optimization strategy achieved a near-linear growth as in Figure 3 is a testimony to its ability to maintain high PE utilisation even in cases of greater parallelism. The behaviour of Latency is such that it shows inverse scaling to the number of PE. In CNN-B workload, inference latency decreases in value between 54.7 ms at 16 PEs and 3.11 ms at 256 PEs. This large setback justifies the power of spatial parallelism and proves that the large method of calculation and not the access to memory are the most significant performance features in the optimised architecture. The trend of the constant rise in the scaling also supports the previously discussed analytical throughput model.

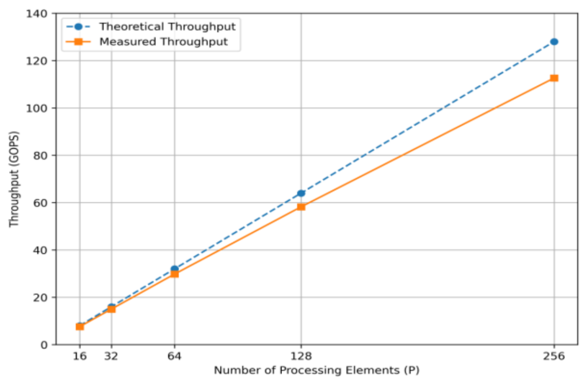

Fig. 3. Throughput scaling of the proposed VLSI accelerator as a function of processing element (PE) count.

Energy efficiency is checked with the comparison of energy used to make one inference among the hardware platforms and is depicted in Figure 4. The comparisons entail a CPU base, GPU base, FPGA accelerator, and the proposed VLSI architecture with the same working conditions. The CPU inferences about 1914.5 mJ and the GPU inferences about 1080 mJ. The FPGA based accelerator uses only 90 mJ per inference of energy. Comparatively, the suggested VLSI architecture has a 10.9 mJ / inference throughput using 256 PEs. Figure 4 indicates that the logarithmic scale is used to emphasise the magnitude of improvement the proposed design will provide compared to general-purpose platforms. This is because it has less off-chip memory traffic, arithmetic is fixed-point which is optimised, and use fixed parallel MAC. The peak performance of the computational unit is equivalent to 112.6 GOPS at 3.5 W, equivalent to about 32.17 GOPS/W, and shows the ability to be used in energy-restricted edge inference systems.

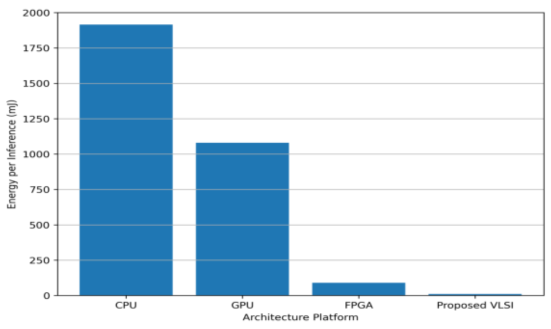

Fig. 4. Energy per inference comparison showing reduced energy consumption of the proposed VLSI architecture relative to CPU, GPU, and FPGA platforms.

The architectural efficiency is also further quantified by post-synthesis area and power analysis. The overall silicon area that is produced by the accelerator under 28 nm CMOS technology is about 6.8 mm 2. The array of processing elements represents approximately 58 percent of the overall area, on-chip SRAM buffers, represent 28 percent, the accumulator tree, takes 8 percent, and control logic represents the remainder 6 percent. This results in a total dynamic power consumption of 3.5 W at 256 PEs with the leakage power constituting about 8 percent of the overall power budget. The architecture has shown a performance of around 16.56 GOPS/mm 2 in terms of area efficiency, based on the throughput and silicon footprint attained. This even distribution of space and energy proves that the given architecture yields high rates of the computation and still does not use much energy. Altogether, the findings in Figures 3 and 4 along with the quantitative analysis of the area and power consumed by the hardware-accelerated VLSI architecture show that it achieves scalable performance, a steep reduction of the latency, and saving significant energy in comparison with the conventional CPU, GPU, and FPGA platforms, and is capable of realistic silicon area and power requirements in the context of deploying edge AI applications.

6. DISCUSSION

The results of weaving the experiment indicate that the proposed VLSI architecture has a near-linear scalability to the growth in the number of processing elements (PEs) as the trends in the throughput presented in Figure 3 confirm. Tightness between the theoretical and measured throughput can be used to assert that the architecture is an efficient way of leveraging spatial parallelism. Nevertheless, as suggested by the slight non-ideal scaling at high count of PE, it means the beginning of memory and interconnects constraints. Though, on-chip buffering and dataflow architecture used reduce much of the reliance on the external memory, it is impossible to eradicate the memory-bound effect entirely as the level of parallelism grows. This has been the case with practical implementations, as accelerators, after all, have memory bandwidth that eventually restricts sustained use. The results of the post-synthesis also show the area-performance trade-off. Adding more PEs in the same proportionately enhances the throughput, however, silicon area and dynamic power consumption increases. Under the existing design, nearly half of all silicon area is taken up by the PE array, which is a representation of the pre-eminence of arithmetic units in the architecture. The attained area efficiency indicates that the design does not have an overwhelming number of overhead control or memory subsystems. However, the scaling beyond 256 PEs would demand a keen consideration of the routing congestion, clock distribution issues and SRAM scaling factor.

The main limitation to the further performance improvement is memory-bound limitations. The higher the PE parallelism, the higher will be the need to have simultaneous activation and weight delivery. In spite of bandwidth pressure alleviation through tiling and data reuse techniques, long-term peak performance is contingent on the capacity of the on-chip memory hierarchy aspect to handle parallel data streams. The issue could be improved in the future by hierarchical buffering, broader memory interfaces, or network-on-chip (NoC) based interconnect to continue to be efficient at higher levels of parallelism. The suggested architecture is especially suitable to edge AI applications. Its fixed-point implementation, lower energy coverage per inference and intermediate silicon area size render it suitable to be deployed in energy-limited systems like embedded systems, smart sensors and mobile AI devices. The specialised VLSI implementation is energy efficient with a significant degree of saving in comparison with CPU and GPU platforms but with deterministic latency behaviour. They are necessary for edge scenarios that have severe power constraints and power thermal conditions, and can only support real time inference workloads. The overall scale of the architecture is positive in terms of its scalability, power efficiency and use of silicon, as well as it will confront realistically achievable memory and routing costs associated with parallel hardware accelerators.

CONCLUSION

This paper offered a scaled hardware-accelerated VLSI hardware platform to run energy-constrained edge applications with minimal energy consumption deep neural network inference. The proposed structure consists of an array of structured processing elements (PE) and pipelined fixed-point multiply accumulate data paths, optimized on-chip memory hierarchy and a coordinated dataflow control mechanism. Analytical throughput model was proposed to describe theoretical scalability and this made it possible to quantitatively verify performance trends with increasing parallelism. The construction was done based on a 28 nm CMOS technology and tested after synthesis with real workload conditions. The experimental findings indicated that throughput was nearly linear with the number of PE with conventional model being close to the analytical model. The architecture with 256 processing elements at 500 MHz produced 112.6 GOPS, and had high utilisation efficiency. The memory-intelligent design proved to be effective in overcoming bandwidth constraints so that the performance could follow a longer curve without hitting the viability limit. Besides being scalable, the architecture produced significant changes in both inference latency and energy efficiency. Energy consumption of the CNN assignment received was minimised to 10.9 mJ/inference, which was considerably lower than the CPU, GPU and FPGA-based systems. The area and power analysis further validated that the design is characterised by an attractive balance of silicon foot, dynamic power, and computational density and provides competitive area efficiency and high performance per watt. In general, the findings confirm that prudently co-engineered computation, memory hierarchy and dataflow scheduling are the key success factors to scalable and energy efficient DNN acceleration in custom VLSI implementations. The architecture is quite suitable to use in edge AI systems where deterministic latency and power consumption are important. Future research on the work can involve support of mixed-precision arithmetic to bring even less energy usage, adoption of adaptive voltage and frequency scaling mechanisms, and consideration of hierarchical or network-on-chip connectivity to allow higher scalability of the current PE design. Moreover, it can also be used to incorporate load-scheduling dynamics, which might be implemented using the runtime scheduling mechanisms, to be further useful in the case of various neural network models.

REFERENCES

1. Albericio, J., Judd, P., Hetherington, T., Aamodt, T., Jerger, N. E., &Moshovos, A. (2016). Cnvlutin: Ineffectual-neuron-free deep neural network computing. *ACM SIGARCH Computer Architecture News, 44*(3), 1–13.
2. Chen, T., Du, Z., Sun, N., Wang, J., Wu, C., Chen, Y., & Temam, O. (2014). DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. *ACM SIGARCH Computer Architecture News, 42*(1), 269–284.
3. Chen, Y. H., Krishna, T., Emer, J. S., & Sze, V. (2016). Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits, 52*(1), 127–138.
4. Chen, Y. H., Yang, T. J., Emer, J., & Sze, V. (2019). Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems, 9*(2), 292–308.

5. Du, Z., Fasthuber, R., Chen, T., Ienne, P., Li, L., Luo, T., … Temam, O. (2015). ShiDianNao: Shifting vision processing closer to the sensor. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture* (pp. 92–104).
6. Han, S., Liu, X., Mao, H., Pu, J., Pedram, A., Horowitz, M. A., & Dally, W. J. (2016). EIE: Efficient inference engine on compressed deep neural network. *ACM SIGARCH Computer Architecture News, 44*(3), 243–254.
7. Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., … Yoon, D. H. (2017). In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture* (pp. 1–12).
8. Kwon, H., Samajdar, A., & Krishna, T. (2018). MAERI: Enabling flexible dataflow mapping over DNN accelerators via reconfigurable interconnects. *ACM SIGPLAN Notices, 53*(2), 461–475.
9. Liu, D., Chen, T., Liu, S., Zhou, J., Zhou, S., Temam, O., … Chen, Y. (2015). PuDianNao: A polyvalent machine learning accelerator. *ACM SIGARCH Computer Architecture News, 43*(1), 369–381.
10. Parashar, A., Rhu, M., Mukkara, A., Puglielli, A., Venkatesan, R., Khailany, B., … Dally, W. J. (2017). SCNN: An accelerator for compressed-sparse convolutional neural networks. *ACM SIGARCH Computer Architecture News, 45*(2), 27–40.
11. Sharma, H., Park, J., Suda, N., Lai, L., Chau, B., Kim, J. K., … Esmaeilzadeh, H. (2018). Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural networks. In *Proceedings of the 45th Annual International Symposium on Computer Architecture* (pp. 764–775).
12. Sze, V., Chen, Y. H., Yang, T. J., & Emer, J. S. (2017). Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE, 105*(12), 2295–2329.