Design of a Low Power and Area Efficient Bfloat16 based Generalized Systolic Array for DNN Applications

Ankita Tiwari
Dept. of Electronics and Electrical
IIT Guwahati
Guwahati, India
ankitatiwari@iitg.ac.in

Saras Mani Mishra
Dept. of Electronics and Electrical
IIT Guwahati
Guwahati, India
m.saras@iitg.ac.in

Prithwijit Guha
Dept. of Electronics and Electrical
IIT Guwahati
Guwahati, India
pguha@iitg.ac.in

Pidanic Jan
Department of Electrical Engineering
University of Pardubice
Pardubice, Czech Republic
jan.pidanic@upce.cz

Zdenek Nemec
Department of Electrical Engineering
University of Pardubice
Pardubice, Czech Republic
zdenek.nemec@upce.cz

Gaurav Trivedi
Dept. of Electronics and Electrical
IIT Guwahati
Guwahati, India
trivedi@iitg.ac.in

2022 32nd International Conference Radioelektronika (RADIOELEKTRONIKA) | 978-1-7281-8686-3/22/$31.00 ©2022 IEEE | DOI: 10.1109/RADIOELEKTRONIKA54537.2022.9764899

***Abstract*—Nowadays demand for artificial intelligence (AI) enabled mobile platforms is increasing. From healthcare services to defense and from remote to urban area, there is a huge demand of secured and power efficient devices. The performance of these platforms can be enhanced by providing an efficient compute engine. These compute engines perform a huge amount of matrix operations. The most popular choice for large matrix computation is a systolic array. In general, the systolic array performance degrades for the large input matrices, due to the trade off between resource utilization and computation delay. To address this issue, we need a systolic array with a control unit to re-configure the array according to the requirement of the computation. Computation array can be further improved by handling the negative weights and reduce the MAC operations. In this paper, we proposed a generalized bfloat16 based systolic array in which the sign of the partial sum (PS) is predicted before computation. The PS sign aids in network pruning which enhances system performance. The proposed system is implemented on a Virtex-7 FPGA board and it performs $2.21\times$ and $4.19\times$ better in terms of area and power compared to single-precision based systolic array.**

***Index Terms*—Systolic, Floating-point number system, Bfloat16, DNN**

I. Introduction

In the era of artificial intelligence, a model needs to be trained with a huge dataset. The training and inferencing of a deep neural network (DNN) is generally performed in a graphics processing unit (GPU) because they are faster and more efficient than a CPU. This is due to its parallelism [1]. Nowadays, a tensor processing unit (TPU) is employed for computations, it is a custom ASIC solution available for DNN applications [2]. TPUs are designed for servers in a datacenter as they consume more energy.The DNN application needs to perform matrix computations (either multiplication or convolution operation) and systolic processors can be employed for this purpose [3]. The network of processing elements (PEs) forms a systolic array and it is fully pipelined and uniform. These array architectures reduce the area of a chip by communicating data simultaneously among PEs. However, a lack of proper dataflow reduces its computation speed.

The systolic arrays are employed for matrix computation [4]. The convolution computation module can be converted to a matrix multiplication unit by an image to column (im2col) algorithm [5], shown in Figure 1. The scheduling of PEs in systolic array is determined by data mapping and its flow. The most commonly used dataflow technique is output stationary (OS) [2] [4], since it reuses the output data during computation. The OS dataflow is shown in Figure 1. The output matrix is a fixed array and both input matrices are shifted horizontally and vertically among PE tiles [4]. This array is used in traditional convolution applications (such as convolution neural network). The dimensions of the PE array of typical convolution layers are much larger than systolic array size. Due to which, proper tiling and processing is required to fully exploit the systolic array [4].

However, there are a few special cases of convolution which are complex to compute. The discontinuous feature map and insufficient filter dimensions (input of the PEs) is responsible for computation complexity and this leads to serious performance loss and the systolic array shrinks to a 1-D array only (or matrix to vector multiplication). These issues can be addressed by designing a systolic processor, with a proper dataflow and data mapping, to exploit all the PEs of a systolic array.

As we know in the era of artificial intelligence, a DNN provides better accuracy. Therefore, a DNN models must be trained with a large amount of data which needs billions of

978-1-7281-8686-3/22/$31.00 ©2022 IEEE

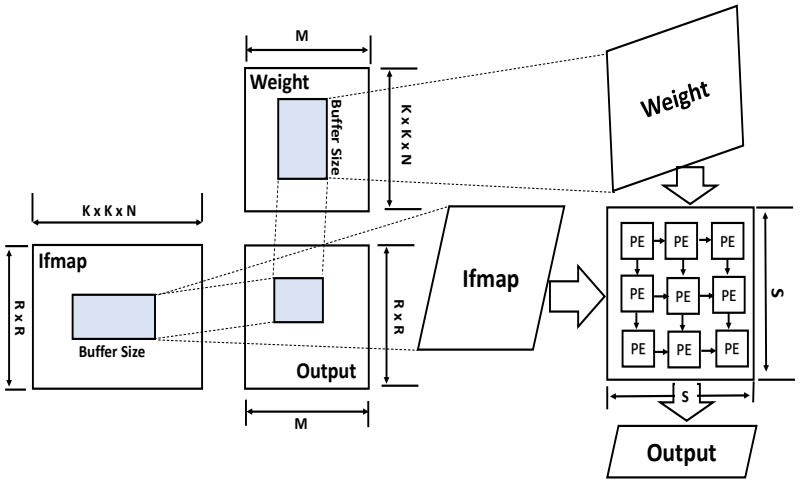

Fig. 1. Image to Column Algorithm on Systolic Array.

computations and consumes hundreds of megabytes of storage. To date, there are various state-of-the-art [6] [7] [8] proposed for accelerating DNN models. The computation cost can be reduced by exploiting pruning, which assigns zero to the most negative weights [9].

As the rectified linear unit (*ReLU*) is the most popular activation function used in hidden layers of DNN models. This function makes the negative outputs zero and introduces nonlinearity to the layer. During hidden layer computations, if the system pre-identifies that the output of a node is going to be negative (which will pass through ReLU), zero is directly assigned to the output without computation. This process can greatly reduce both computation cost and delay of the systolic array. Similar for the *tanh* activation function, when the outputs are smaller than -1, the same concept is applicable. Therefore, the pre-detection of a negative result of MAC can be ignored due to the use of activation function, and this property reduces more MAC operations and memory access. In the proposed unit, we introduced a negative detection unit, which will ignore the negative value after passing through the activation function.

In this paper, we propose a bfloat16 based systolic array architecture. We tried to improve the resource utilization of PEs by modifying it. The following points are addressed in this paper.

- We proposed a novel bfloat16 based floating-point systolic array architecture for DNN applications.
- A shared and global concept for storage and control unit is introduced, which enables the generalized property of a systolic processor.
- A negative output detection is introduced to reduce MAC operations and memory access.

The rest of the paper is explained as follows: the number system is explained in the related work section II, the proposed architecture is explained in section III. Section IV and V explains the results with by presenting an application of proposed systolic array and concludes the proposed work.

II. Related Work

While designing a VLSI architecture, we have to fit all the components and memory units in a fixed area. As the memory is limited, the infinite precision cannot be stored using a binary or fixed-point. To address this issue, the floating-point representation can be exploited. A floating-point number is divided in three parts: *base*, *significand* and *exponent*. It is represented in the format of $significand \times base^{exponent}$.

In literature, there is a standard floating-point representation popularly known as IEEE754 representation. There are many types of IEEE754 representations available, which variously effect the range, bit-width of exponent and mantissa. The most commonly used floating-point representation by computer architects is a single-precision floating-point representation. It is a $32-bit$ representation, which is divided into $1-bit$ *sign*, $8-bits$ *exponents*, and $23-bits$ assigned for *mantissa*.

Recently, there has been a variant named the brain floating-point (bfloat16), of single-precision, proposed by Google Brains. It is a research group at Google for artificial intelligence [10]. Unlike single-precision, it is a 16-bit representation containing $1-bit$ *sign*, $8-bit$ *exponents*, and $7-bits$ reserved for *mantissa*. The range of bfloat16 representation is the same as single-precision i.e. $\sim 1e^{-38}$ to $\sim 3e^{38}$. The value of single-precision and bfloat16 number is calculated through equation 1.

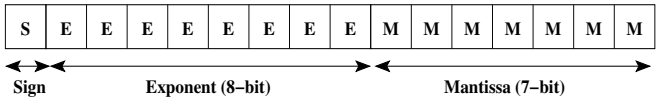

Fig. 2. Bfloat16 Number Format

$$FinalValue = (-1)^S \times 2^{E-127} \times (1.M) \quad (1)$$

Since the exponent bit-width of both the number systems is the same, there are only a few mantissa bits eliminated for conversion. Bfloat16 to single-precision conversion is performed by simply adding sixteen zeros at the LSB of mantissa and vice-versa. Since the bit-width of bfloat16 is smaller than single-precision32, it has lower implementation complexity and faster execution speed.

III. Proposed Architecture

In this paper, we have proposed a re-configurable systolic array. The block diagram is shown in Figure 3. We have modified the multiply and accumulate unit (MAC) proposed in [11]. We introduced a global buffer to store the input features (Ifmap) and weights (w) from DRAM, to reduce the memory read and write operations of the processor. In the following sections, we briefly discussed about each unit with a corresponding block diagram.

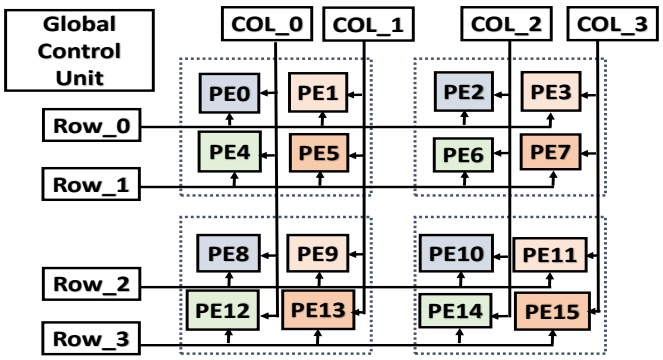

Fig. 3. Systolic Array Architecture for Deep Neural Network Applications.

A. PE Specifications

The proposed design is based on a 16-bit bfloat16 floating-point representation. The control unit and global buffer is separated, and shared among each processing element (PE). The block diagram of PE is shown in Figure 4. Each PE contains an address decoder to fetch the rows of *Ifmap* and *w* from the global buffer, as per the operation. There is a 16-bit internal register available in each PE, to store the partial sum (PS). The partial sum created by another PE is shared via the nearest port (p0/p1) and this setup is called double-PE (explained in subsection III-C). We employed a 16-bit floating-point multiplier and adder for computation. An acknowledgement signal (ack) is generated by each PE to the global control unit, after completion of processing within MAC.

B. Global Buffer Specifications

This global buffer has twelve 8×16-bit Ifmap storages, four 4×16-bit zero buffer storages, three 16×16-bit weight storages, and a 8×9-bit for sign bits. The total size of global buffer is $144MB$. In the proposed module, *Ifmap* and *w* rows are stored in a global buffer from DRAM. A block diagram of the global buffer is shown in Figure 5. It is designed as a dual

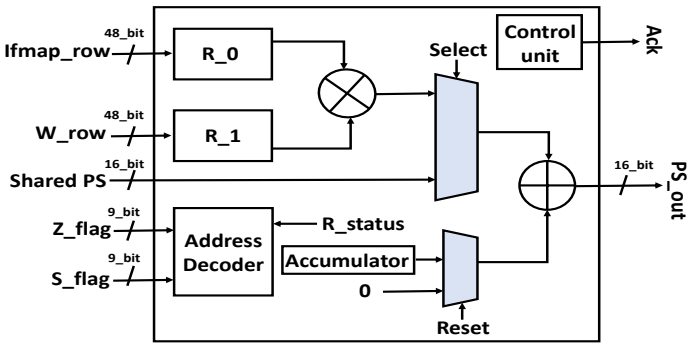

Fig. 4. Architecture of Processing Element.

port register bank. *Ifmap* storage is divided in two separate groups because it is required to maximize local data reuse [9]. The group of six 8×16-bit buffers are divided in two subgroups of three 8×16-bit registers. The Ifmap storage is employed as a temporary memory element for *Ifmap* rows of the channel. The 4×16-bit zero buffer storage is employed for zero flags to store even and odd channels in the *Ifmap* storage. The buffer elements have even and odd channel values as shown in Figure 5. The acknowledgement from each PE directs the main control unit to receive the input from the global buffer.

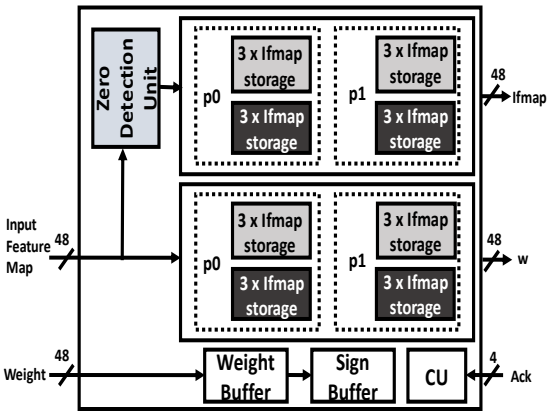

Fig. 5. The Proposed Global Buffer.

The control unit only writes the data into the global buffer when it gets the acknowledgement signal from the PE. It provides a signal to validate the storage of a *Ifmap* row from the address provided by the address decoder of the PE. It is a $12 - bit$ storage for *Ifmap* and *w* ports and there are three elements in each port.

C. Double-PE

Figure 6 shows the sub-units of the proposed double-PE. We called it double-PE because it contains two PEs and also contains multiplexers, two storage elements and a shared control unit. A storage unit and a MAC, combinedly called as a single PE, is shown by the red box in Figure 6. The storage

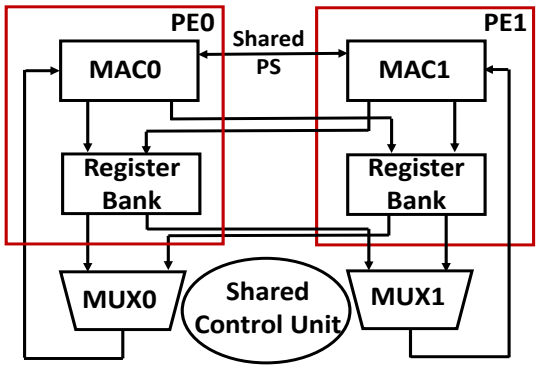

Fig. 6. The Architecture of Proposed Double-PE.

elements are connected to MAC through multiplexers. The control unit directs the data distribution and storage among PEs. The shared storage unit is accessed by PE_0 when PE_1 is at rest and PE_1 when PE_0 is at rest. The global control unit provides the feedback to a shared control unit about the state of PE (rest or active). The shared control unit notifies the storage unit to reserve PE_0 for computation of odd rows. It also connects the PE_1 and storage through the multiplexer to compute the odd rows of *Ifmap* and *w*.

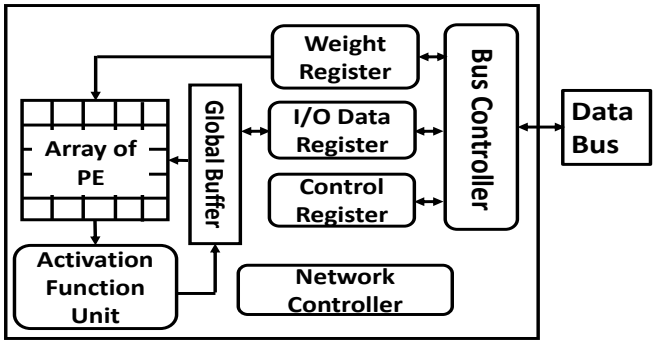

Fig. 7. Application of Systolic Array.

D. Dataflow

In the proposed module, the PE clusters are divided into four groups, as shown in Figure 3. Every group contains four PEs and distinguished by the dotted line shown. One PE from each group (shown by same color in figure) share their storage element for optimum speedup. The PEs available in other clusters coordinate with each other to perform computations and data transfer. Cross-wise sharing allows the storage element and provides a chance to share data with both row and column clusters of each group for computation. As shown in Figure 3, all the rows and columns are connected to each other.

Each PE cluster (shown in Figure 3) computes the result at the same time, which reduces the delay and enhances the matrix computation. Furthermore, the *w* can be stored in a global buffer (or offline) to reduce memory operations, which will reduce the power consumption. It is worth mentioning that *Ifmap* and *w* have the minimum negative weights loaded during initialization.

E. Architecture of Systolic Processor

The proposed systolic array contains sixteen PE clusters and one row and column storage in terms of the global buffer. There is a 2×2 array of PE available in each cluster. The PEs of each column calculate input for a different feature output. There are two elements available in each *Ifmap* and *w* row inputs. The feature rows with more than two elements are divided into rows of two elements and zero padding in the case of missing values. Every shared storage unit contains 6kB and 1.5kB SRAM for storing *Ifmap* and *PS* values, respectively. Additionally, the sign of *w* in each row is held in a separate 1.2kB SRAM buffer and shared among columns. The *Ifmap* and *w* rows are connected via row stationary (RS) dataflow [12]. They re-use the diagonal value in the PE array during matrix computation. Further, the PE of each cluster is assigned with a unique identification, which helps the control unit to distinguish between the ideal or active states of the PE. Each column of the systolic array contains a shared control unit to coordinate among the PE, *PS*, and shared buffer.

TABLE I
RESOURCE UTILIZATION SYSTOLIC ARRAY

Resource	Single-precision		Bfloat16	
	Available	Utilization	Available	Utilization
LUT	433200	6752	433200	3459
Slice Reg	11120	5399	11120	3670
FF	15311	37	866400	16
IO	850	65	850	51
BUFG	32	7	32	1
Total Resources	1311602	12260	1311602	7197
Comparative Utilization	**1.70X**		**X**	

IV. RESULTS OF SYSTOLIC ARRAY

In general, systolic processors are employed in matrix computation. In this section, we demonstrated the matrix computation of a neural network for a verification purposes. We used a small wine dataset [13] for a multilayer neural network model. It is a three-class classification problem. It has 12 features in the dataset and 178 instances. We have used 80% of the dataset for training and 20% for inference. The weights are generated by training the model in Spyder using python. In this section, we tested the performance of systolic array by providing inputs (weights (w) matrix $m \times n$ and features (Ifmap) matrix $n \times m$) and verified the result. Input matrices are given in the form of neural network with two hidden layers, one input and output layer. The network has 12 nodes in the input layer followed by 80 and 50 nodes in first and second hidden layers, respectively. The input layer computes a 1×12 and 12×80 matrix and after ignoring all negative values the result is fed as input to the hidden layer. The first hidden layer computes the matrices of dimensions 1×80 and 80×50, and second hidden layer computes the

dimensions 1×50 and 50×3. As the three class (output vector dimension is 1×3) classification problem, it has three nodes in the output layer.

TABLE II
PERFORMANCE COMPARISON

Model	Parameter	Single-precision	Bfloat16	Improvement
Two layer NN	Test Accuracy (%)	95.55	94.23	1.32% ↓↓
	Area (mm^2)	18240.821	8240.6237	2.21×↑↑
	Power (uW)	84132.356	20052.356	4.19×↑↑

The proposed application is shown in Figure 7. The systolic array is employed for matrix multiplication, i.e. for the proposed application there will be three matrix multiplications which need to be performed. The result of the proposed systolic array is compared with software simulation results for a performance check. The accuracy of the hardware realization of the systolic array is 94.23% which is close to software simulation. The hardware model parameter of the proposed application is shown in Table II. The proposed unit consumes $2.21\times$ and $4.19\times$ less area and power compared to single-precision based implementation. The resource utilization of the proposed model is shown in Table I. Significantly, the bfloat16 based systolic array consumes $1.70\times$ less resources compared to the single-precision based systolic array.

V. CONCLUSION

In this paper, a configurable systolic array is proposed. This module can be used for a real-time matrix computation in DNN applications. During software simulation, it is analyzed that bfloat16 based DNN models perform better compared to fixed-point based models. The proposed bfloat16 based systolic array provides better resource utilization and is 70% more efficient than the single precision based computation array. The bfloat16 based systolic array provides 98.61% accuracy and $2.21\times$ and $4.19\times$ less area and power compared to single-precision based systolic array implementation. This makes the proposed bfloat16 based systolic array a perfect candidate to be utilized in DNN based applications.

ACKNOWLEDGEMENT

The work has been supported from the programme INTER-EXCELLENCE (LTAIN19100) funds of the Ministry of Education, Youth and Sports, Czech Republic, project LTAIN19100 Artificial Intelligence Enabled Smart Contactless Technology Development for Smart Fencing. This support is very gratefully acknowledged.

REFERENCES

[1] G.-B. D. L. Inference and B. D. Learning, "A performance and power analysis," *NVidia Whitepaper, Nov*, 2015.

[2] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th annual international symposium on computer architecture*, 2017, pp. 1–12.

[3] S. Y. Kung, "Vlsi array processors," *Englewood Cliffs*, 1988.

[4] A. Samajdar, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna, "Scale-sim: Systolic cnn accelerator simulator," *arXiv preprint arXiv:1811.02883*, 2018.

[5] K. Yanai, R. Tanno, and K. Okamoto, "Efficient mobile implementation of a cnn-based object recognition system," in *Proceedings of the 24th ACM international conference on Multimedia*, 2016, pp. 362–366.

[6] Y. Choi, D. Bae, J. Sim, S. Choi, M. Kim, and L.-S. Kim, "Energy-efficient design of processing element for convolutional neural network," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 64, no. 11, pp. 1332–1336, 2017.

[7] K. Guo, L. Sui, J. Qiu, J. Yu, J. Wang, S. Yao, S. Han, Y. Wang, and H. Yang, "Angel-eye: A complete design flow for mapping cnn onto embedded fpga," *IEEE transactions on computer-aided design of integrated circuits and systems*, vol. 37, no. 1, pp. 35–47, 2017.

[8] S. Zhou, Q. Guo, Z. Du, D. Liu, T. Chen, L. Li, S. Liu, J. Zhou, O. Temam, X. Feng *et al.*, "Paraml: A polyvalent multicore accelerator for machine learning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 9, pp. 1764–1777, 2019.

[9] M. Asadikouhanjani and S.-B. Ko, "A novel architecture for early detection of negative output features in deep neural network accelerators," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 67, no. 12, pp. 3332–3336, 2020.

[10] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, 2016, pp. 265–283.

[11] A. Tiwari, G. Trivedi, and P. Guha, "Design of a low power bfloat16 pipelined mac unit for deep neural network applications," in *2021 IEEE Region 10 Symposium (TENSYMP)*. IEEE, 2021, pp. 1–8.

[12] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE journal of solid-state circuits*, vol. 52, no. 1, pp. 127–138, 2016.

[13] C. Blake, "Uci repository of machine learning databases," *http://www. ics. uci. edu/˜ mlearn/MLRepository. html*, 1998.