Accelerating Attention Mechanism on FPGAs based on Efficient Reconfigurable Systolic Array

WENHUA YE, College of Information Science and Engineering, Hunan University, China and China Electronics Technology Group Corporation 36th Research Institute, China
XU ZHOU, College of Information Science and Engineering, Hunan University, China
JOEY ZHOU, Centre for Frontier AI Research, A*STAR, Singapore
CEN CHEN, Shenzhen Institute, Hunan University, China
KENLI LI, College of Information Science and Engineering, Hunan University, China

Transformer model architectures have recently received great interest in natural language, machine translation, and computer vision, where attention mechanisms are their building blocks. However, the attention mechanism is expensive because of its intensive matrix computations and complicated data flow. The existing hardware architecture has some disadvantages for the computing structure of attention, such as inflexibility and low efficiency. Most of the existing papers accelerate attention by reducing the amount of computation through various pruning algorithms, which will affect the results to a certain extent with different sparsity. This paper proposes the hardware accelerator for the **multi-head attention (MHA)** on **field-programmable gate arrays (FPGAs)** with reconfigurable architecture, efficient systolic array, and hardware-friendly radix-2 softmax. We propose a novel method called **Four inputs Processing Element (FPE)** to double the computation rate of the **data-aware systolic array (SA)** and make it efficient and load balance. Especially, the computation framework is well designed to ensure the utilization of SA efficiently. Our design is evaluated on a Xilinx Alveo U250 card, and the proposed architecture achieves 51.3×, 17.3× improvement in latency, and 54.4×, 17.9× energy savings compared to CPU and GPU.

CCS Concepts: • **Computer systems organization** → **Systolic arrays**; System on a chip; • **Computing methodologies** → *Machine translation;*

Additional Key Words and Phrases: Transformer, attention, FPGA, reconfigurable systolic array, softmax, Accelerator

This work was supported in part by the Key-Area Research and Development Program of Guangdong Province under Grant 2021B0101190004, the Natural Science Foundation of Hunan Province under Grant 2022JJ30009, the NSFC under Grants 62172146 and 62172157, the Open Research Projects of Zhejiang Lab under Grant 2021KD0AB02, the Cultivation of Shenzhen Excellent Technological and Innovative Talents (Ph.D. Basic Research Started) RCBS20200714114943014, and the Basic research of Shenzhen Science and technology Plan under Grant JCYJ20210324123802006.
Authors' addresses: W. Ye, College of Information Science and Engineering, Hunan University, 2 Lushan South Road, Changsha, Hunan, 410082, China and China Electronics Technology Group Corporation 36th Research Institute, 387 Hongxin Road, Jiaxing, Zhejiang, 314033, China; X. Zhou (corresponding author) and K. Li, College of Information Science and Engineering, Hunan University, 2 Lushan South Road, Changsha, Hunan, 410082, China; J. Zhou, Centre for Frontier AI Research, A*STAR, 1 Fusionopolis Way, #08-01 Connexis, 138632, Singapore; C. Chen, Shenzhen Institute, Hunan University, Shenzhen High-tech Industrial Park South District Virtual University Park, Shenzhen, Guangdong, 518000, China.

1539-9087/2023/11-ART93 $15.00
https://doi.org/10.1145/3549937

ACM Reference format:
Wenhua Ye, Xu Zhou, Joey Zhou, Cen Chen, and Kenli Li. 2023. Accelerating Attention Mechanism on FPGAs based on Efficient Reconfigurable Systolic Array. *ACM Trans. Embedd. Comput. Syst.* 22, 6, Article 93 (November 2023), 22 pages.
https://doi.org/10.1145/3549937

1 INTRODUCTION

The Transformer has made great progress in **natural language processing (NLP)** [5, 25, 29], computer vision [35, 37, 42], and other fields in recent years. Many Transformer-based models have emerged, such as Transformer [29], BERT [17], Transformer-XL [4], Routing Transformers [26], and the like. One of their remarkable features is the application of the attention mechanism, which is significantly different from the traditional **Convolutional Neural Network (CNN)** and **Recurrent Neural Network (RNN)**.

In Transformer, the model contains a multi-headed self-attention mechanism. The attention mechanism includes multiple matrix multiplications, which increase squarely with the sequence length [20]. ELSA [10] shows that self-attention accounts for a significant portion of the runtime across many existing self-attention-oriented **Neural Network (NN)** models (BERT, RoBERTa, ALBERT, BERT4Rec) from about 38% to 64% when the sequence length increases to 4x. Some works [9, 10, 14, 20, 32] pointed out that the attention mechanism is an expensive operation because of its intensive matrix computations and complicated data flow. Unfortunately, attention runs extremely slow on general-purpose platforms such as GPUs and CPUs.

FPGA is widely used to accelerate deep learning algorithms for its high parallelism and low latency [8]. There are quite a lot of works focusing on the parallelizing calculation to accelerate CNN [23, 30] and **Graph Convolutional Network (GCN)** [6, 40, 41]. Recently, there are some works [9, 14, 20, 21, 32] that have built FPGA or **Application Specific Integrated Circuit (ASIC)** hardware accelerators for Transformer and achieved a certain success. However, these works accelerate attention by reducing the amount of computation through various pruning algorithms, which will affect the results to a certain extent with higher sparsity. Especially, different data sets have different sparsity, which brings a great challenge for these pruning algorithms.

The efficient systolic array is another promising solution [36]. It has very high parallelism and is suitable for acceleration on FPGA hardware. Literature [21] uses logic resources to implement SA, which wastes **digital signal processing (DSP)** resources that can calculate at high speed accordingly, so the application can only run on a low clock frequency. It is easy to cause timing violation problems with the high resource occupancy rate. However, most Transformer FPGA accelerators [14, 21, 24] use Xilinx **High-Level Synthesis (HLS)** tools for implementation. This tool can automatically compile the code of the C++ language into netlist to implement functions. Although it is more efficient than **hardware description language (HDL)** in implementation time, it cannot control some FPGA resources to realize functions efficiently, such as DSP resources on Xilinx FPGA. Literature [12] presents an effective **single-instruction multiple-data (SIMD)** 3-dimension revolution engine designed for FPGA through mapping two **multiply and accumulate (MAC)** operations within a single computing unit (e.g., DSP slice). Therefore, we introduce **double MAC (DMAC)** ideas to SA and optimize them simultaneously.

To the best of our knowledge, this is the first architecture that efficiently applies DSP48 slices for SA to accelerate the attention and supports dynamic reconfiguration of SA to adapt to input data with different bit widths.

This paper aims to provide an attention accelerator with an efficient reconfigurable systolic array using FPGA. The main contributions of this work can be summarized as follows:

- We design a new computation framework to ensure the high hardware utilization of the SA and high-speed data throughput with DDR. The SA is data-aware and can reconfigure between two forms with the same DSP to make the systolic array efficient and load balance.
- The architecture in this paper is a co-design of software and hardware, which is composed of a two-level scheduler and a two-level state machine. It can flexibly call the operation of the systolic array by choosing the two-level state machine in the second-level scheduler to dynamically configure the calculation process in FPGA to adapt to different sequence lengths and embedding sizes.
- We propose a complete flow of multi-head attention scheme and improve the architecture of the attention after concatenation, which means at the end part of attention, we send the concatenated data to multi-head **systolic arrays (SAs)**. This architecture effectively uses the parallel calculation of all two SAs in the multi heads and reduces the calculation delay.
- We propose a novel **processing element (PE)** called FPE to double the computation rate of SA by packing four multiply, four accumulate, and four addition operations into two DSP slices of FPGA.
- We introduce a specialized method for scaled masked-softmax, without the exponentiation calculation, which is highly optimized to become more hardware-friendly.
- We implement the proposed techniques on different hardware for further comparison of performance and energy consumption. Experimental results show that the FPGA hardware design enables a 51.3×, 17.3× improvement in latency, and 54.4×, 17.9× energy savings compared to CPU and GPU.

The organization of the work is as follows. Section 2 gives a brief review of the attention mechanism in Transformer, explains the importance of accelerating the attention and the advantage of using SA on FPGA. Section 3 presents the analysis of calculation in the attention mechanism. Section 4 proposes the hardware architecture design. Experimental results are given in Section 5. Section 6 shows the related work. Section 7 provides the overall conclusion for the hardware design and experiments.

2 BACKGROUND AND MOTIVATION

2.1 The Model Architecture of the Attention in Transformer

The architecture of the standard Transformer is described in Figure 1 [29], containing an encoder and a decoder, where all the encoder and the decoder layers are composed of three kinds of Blocks, the MHA block, the **feed-forward network (FFN)** block, and the Add & Norm block. In this paper, we mainly accelerate the MHA block calculation. However, the FFN block can be accelerated in our architecture because its main calculation is composed of two linear layers.

Figure 2 shows the structure of the MHA block. An MHA block has h "Attention Heads", and the input of each Head is the same as the input of the Block, including three tensors: V (values), K (keys), and Q (queries). The Scaled Dot-Product Attention function in the MHA is defined as:

$$\text{Attention}\ (Q_i, K_i, V_i) = \text{softmax}\left(\text{Mask}\left(\frac{Q_i K_i^T}{\sqrt{d_k}}\right)\right) V_i \tag{1}$$

$$Q_i = X^* W_q + B_q, K_i = X^* W_k + B_k, V_i = X^* W_v + B_v \tag{2}$$

Specifically, Q_i, K_i, and V_i are linear transformations applied on the temporal dimension of the input sequence X with Weight and Bias. The parameter $d_k = d_{model}/h$ is the dimension of Q_i and K_i, d_{model} is the hyperparameter of embedding layer size.

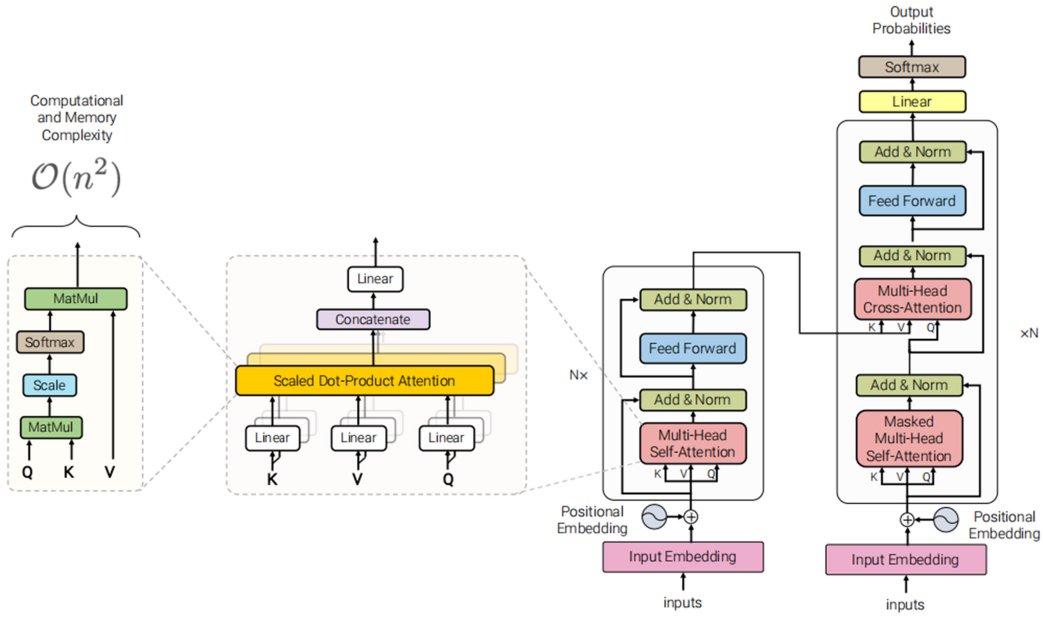


Fig. 1. The architecture of the standard Transformer model [29].

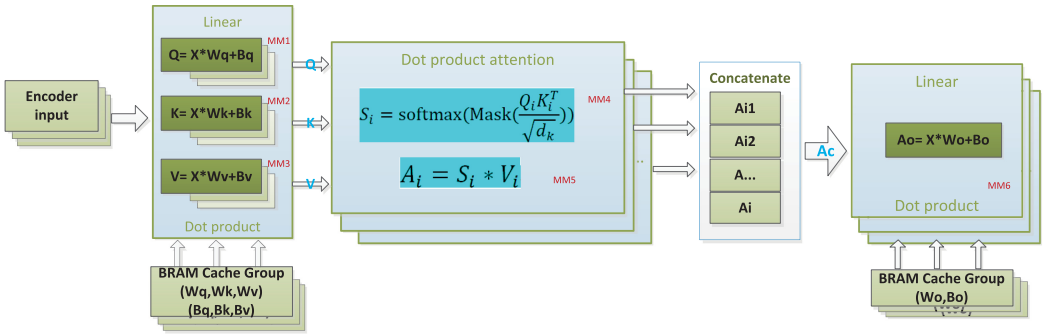


Fig. 2. The structure of the MHA block.

2.2 Motivation

The attention mechanism is widely used in all kinds of Transformer modes, but the calculation of attention includes multiple matrix multiplications. The amount of computation increases squarely with the sequence length. These facts prove the necessity of designing efficient hardware accelerators for the MHA. Implementation of efficient SA on FPGA is a promising solution. The FPGA implementation of most articles adopts HLS [14, 21, 24]. For the dual SAs architecture in this paper, there are data interactions and synchronous control between them, and it is better to use HDL. In addition, it is difficult to use HLS for the precise control of delay and efficient utilization of resources, such as the implementation of **double multiply-accumulation-and-addition (DMAA)** core and **single multiply-accumulation-and-addition(SMAA)**. As far as we know, this is the first architecture that applies DSP48 efficiently for SA to accelerate the attention mechanism and supports dynamic reconfiguration of SA to adapt to input data with different bit widths.

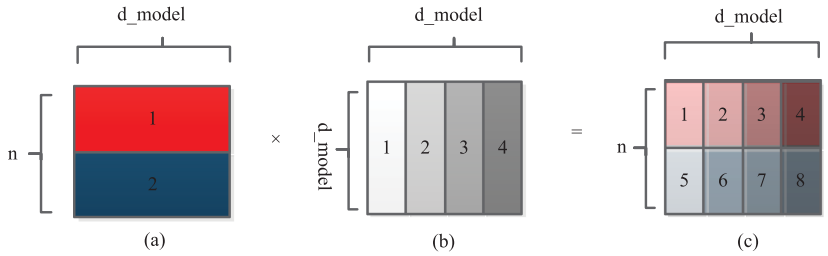


Fig. 3. The division of the matrix multiplication.

3 THE ANALYSIS OF THE ATTENTION MECHANISM

The attention takes three matrices as its inputs, namely the query (Q), the key (K), and the value (V), which are multiplied with the weights of linear layer (W_q, W_k, W_v). The first step of computing attention is to obtain a scoring matrix (S) by multiplying Q and K. The score matrix is often referred to as the attention matrix. Then, the score matrix goes through the softmax function for normalization. The second step is to multiply the normalized score matrix with the value matrix (V) to generate the output of heads, and they will be concatenated together and passed into a dense layer.

3.1 Six Matrix Multiplications

As shown in Figure 2, there are six matrix multiplication (MM1-MM6) computations, all of which can be done by the same SA with different modes. If the row and column of the matrix are larger than the size of SA, we can finish it by multi-loops with the division of the matrix. From Figure 3, the matrix (a) is divided into two rows, the matrix (b) is divided into four columns, and the $matrix(c) = matrix(a) * matrix(b)$ can be calculated with eight loops of SA.

3.2 Linear Function

In SpAtten [32], the linear function ($V * W + Bias$) consists of two parts: the $Y = V * W$ is calculated with SA, then $Y + Bias$ is done by an add tree. However, this calculation can be finished in a SA only by using a special design of the DSP slice.

3.3 Different Quantization

In the first part of the attention in Figure 2, the input matrices (X) of Linear functions (MM1 MM3) are quantized with INT8, and the trainable variable matrices (W, B) are quantized with INT16; the rest of the calculations are all quantized with INT16 to advance the dynamic range. To reuse the same resources to deal with all these calculations, we propose a data-aware SA, which can be configured between ($32Row, 32Col$) SA and (($16Row, 32Col$)) SA with the same utility of resources. Thus in this way, it can save lots of resources and increase the universality of the SA architecture.

3.4 Scaled Masked-Softmax

The exponentiation calculation of softmax is still quite expensive [21]. With the log sum-exp trick and designing algorithmic reduction strategies, a high-speed and low-complexity hardware architecture for the softmax function was proposed [34]. However, the utility of resources in **exponential units (EXPU)** and adder trees will rise rapidly when the number of input increases. To address this problem, we propose the radix-2 softmax method to optimize the calculation to become more hardware-friendly, and the architecture is pipelined.

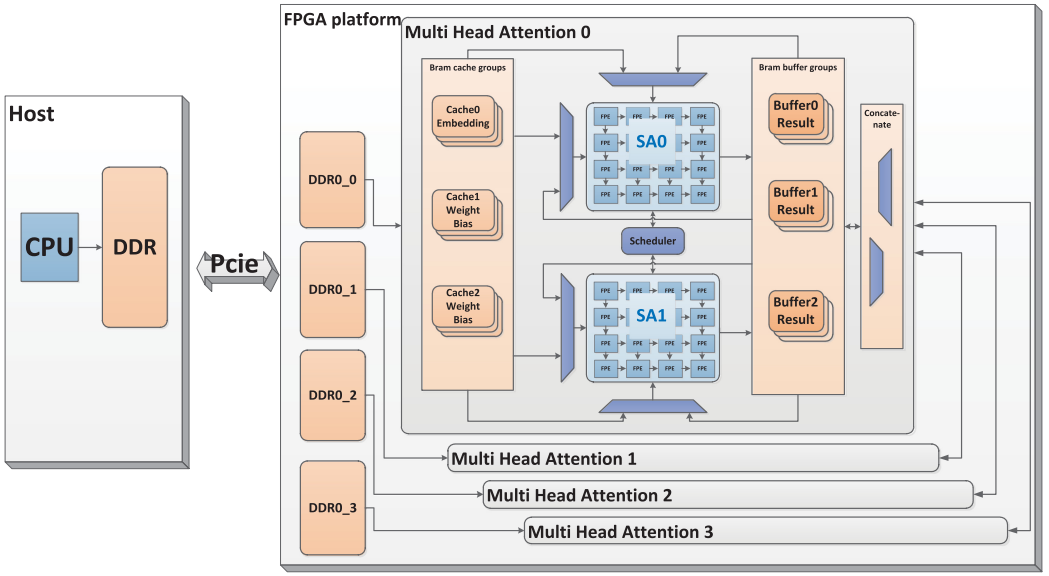


Fig. 4. The top-level architecture.

4 HARDWARE ARCHITECTURE DESIGN

4.1 The Top-Level Architecture

Based on the proposed method, the complete hardware accelerator is designed. The top-level architecture is illustrated in Figure 4.

Firstly, the host generates and quantizes the tokenized sentence, weight, and bias, all of them will be sent to DDRs on the FPGA card through the PCIe interface. Then, the scheduler will fetch the word embedding sequence, weight, and bias from DDR to FPGA on-chip Bitwidth converter cache for calculation, which is a two parts module. The first module is an AXI4 interconnector (containing FIFO and crossbar), it has one 512bit master interface and six 128bit slave interfaces; The second module is the **Block ram (Bram)** cache group, which converts data from 128 bit to 8bit or 16bit, it consists of multi Brams, whose number depends on the row of SA. These Bram cache groups generate the input timing sequence of SA.

There are two SAs in one multi-head attention accelerating kernel, which receive data flow from the Bram cache and Bram buffer groups. The scheduler will decide the input data sources and addresses. The SAs parallel calculate and cache results through the Bram group buffer. The scheduler can achieve computational balance by distributing the calculation to SAs. By the way, the architecture supports SA to execute N cycles to accomplish the matrix multiply. For example, in the $matrix(32, 64) * matrix(64, 64)$, there will be two cycles for a $(32, 32)$ SA to finish this calculation.

Most existing papers design the attention mechanism with multi-head attention from the beginning to the concatenation. However, the calculation is still huge at the end of the attention mechanism, which is $(sequence\ length, d_{model}) * (d_{model}, d_{model})$. In this paper, we propose a complete flow of multi-head attention scheme and improve the architecture of the attention after concatenation, which means at the end part of attention, we send the concatenated data to the row of multi-head kernel's SAs and divide the weight in multi-part to the column of multi-head kernel's SAs. This architecture effectively uses the parallel calculation of all two SAs in the multi heads and reduces the calculation delay. After concatenation, this method is more efficient than one head and one SA.

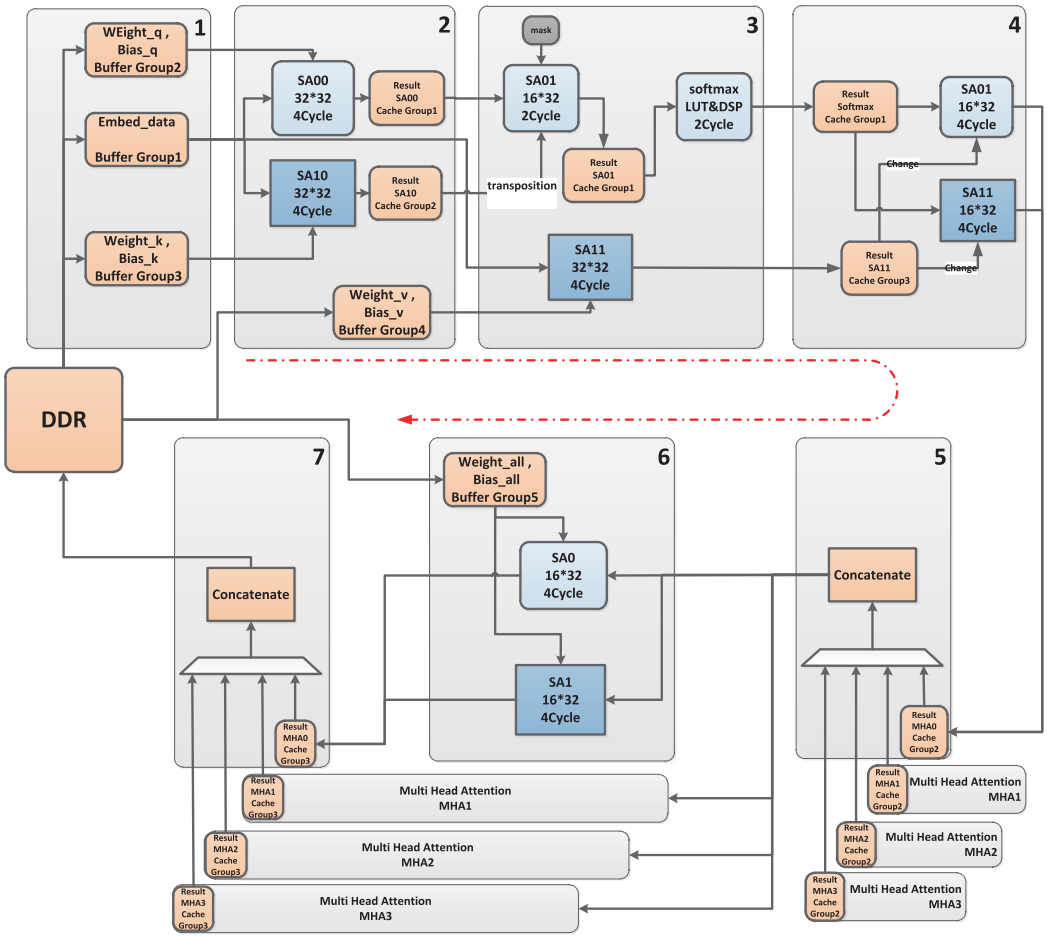


Fig. 5. The full acceleration calculation step.

The whole architecture comprises DDR, bit width conversion Bram cache group, SA group, calculation result Bram buffer group, and softmax. The full acceleration calculation step is divided into seven parts. For example, we set $sequence\ length = 32$, $d_{model} = 512$, $h = 4$. The calculation steps are shown in Figure 5:

(1) The DDR controller reads DDR data to the bit width conversion Bram cache group;

(2) Two SAs calculate the linear of Q and K with four cycles, and the calculation results are buffered by two Bram groups, respectively;

(3) One SA calculates $Q * k$ with two cycles, then softmax, and the Scores are output to the Bram buffer group; The other SA calculates the linear of V with four cycles and outputs the data to the Bram buffer group;

(4) Two SA parallel processing $Scores * V$ with four cycles;

(5) Concatenating of multi-head kernel's results;

(6) To calculate the final linear, use the multi-head attention idea to read the data of step (5) to each multi-head module, and the two SAs of each module are processed in parallel with four cycles;

(7) Merge the multi-head kernel's results and output the calculation results to DDR.

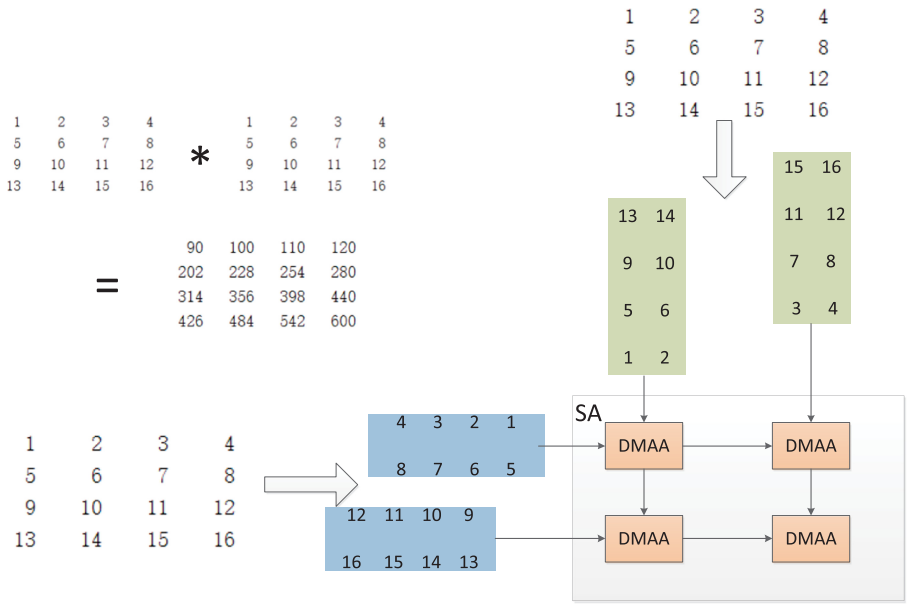


Fig. 6. The architecture of SA.

Table 1. The Data-aware Systolic Array

SA in the architecture	Size of SA	Input bit width	Core	Utilization of DSP
SA00, SA10	(32, 32)	8bit	DMAA	512
SA01, SA11	(16, 32)	16bit	SMAA	512

4.2 Efficient Reconfigurable Systolic Array

In the first stage of attention calculation, the input data are quantized into 8bit. When the calculation results are input to the second stage and subsequent analyses, to have greater dynamic range and calculation accuracy, 16bit datas are input to the SA. Therefore, this paper designs the data-aware SA architecture. The advantage of this SA is its ability to adapt the data, and there is no need to instantiate two types of SA to process input data with different bit widths, which saves a lot of resources.

The architecture of SA is shown in Figure 6, and a matrix multiplication operation of $(4,4)*(4,4)$ is taken as an example. The idea is to use the FPE core to complete the parallel multiplication calculation of two rows and two columns of data simultaneously. And the FPE core can reconfigure and switch between the two forms according to the bit width of the input data because it is internally composed of two DMAA (or SMAA) cores. As shown in Figure 7, they can adapt to the calculation of two rows (8bit data) multiplied with two columns (16bit data) or one row (16bit data) multiplied with two columns (16bit data).

For example, the SA used in this paper can complete a $(32,32)*(32,32)$ matrix multiplication (8 bit input) or a $(16,32)*(32,32)$ matrix multiplication (16 bit input), which is illustrated in Table 1. In addition, SA can be reconfigured between two forms of size with the DMAA(or SMAA) cores, and the utility of DSP is the same.

The FPE can process four multiplications, four accumulations, and four additions in parallel by applying the DMAA core. With this module, four $(V*Weight+Bias)$ calculations can be completed without additional add trees.

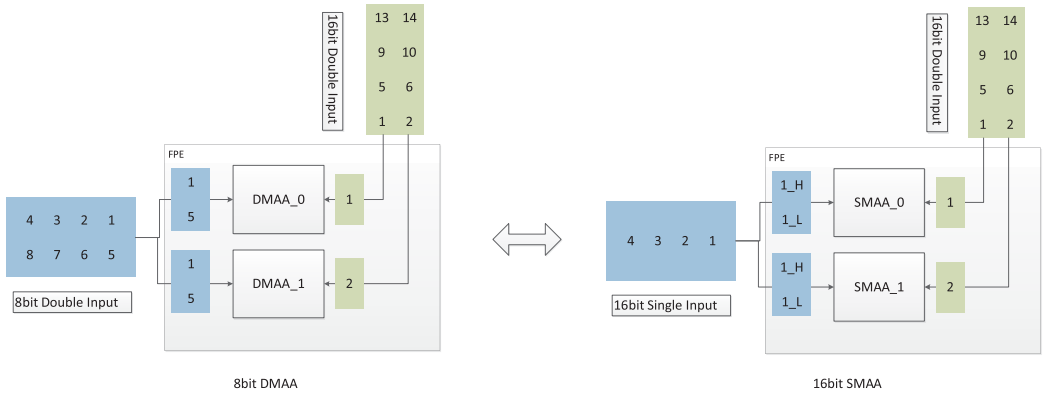


Fig. 7. The architecture of FPE.

Xilinx gives an example of a systolic array based on HLS (https://github.com/Xilinx/Vitis_Accel_Examples/tree/2020.2/cpp_kernels/systolic_array). For a fair comparison, we set MAX_SIZE(the parameter of systolic array size) to 32. The synthesis command [config_op mul–imp] is set to [dsp], which specifies HLS to use DSP to realize multiplication. After the HLS synthesis compilation was completed, we found DSP48 utilization was 1024. So the DSP48 resources of this paper can be saved by half. In addition, the directive of dataflow is added to the module to reduce its latency. However, the SA in this paper can speed up by 1.1x compared to HLS SA. The most important thing is that the HLS SA does not have the function of dynamic reconfiguration, which means it can not apply DSP48 efficiently. To meet the needs of different functions, the HLS SA module should be designed with the maximum bits or organized into multiply modules, which will cause a waste of resources.

4.3 DMAA and SMAA Cores

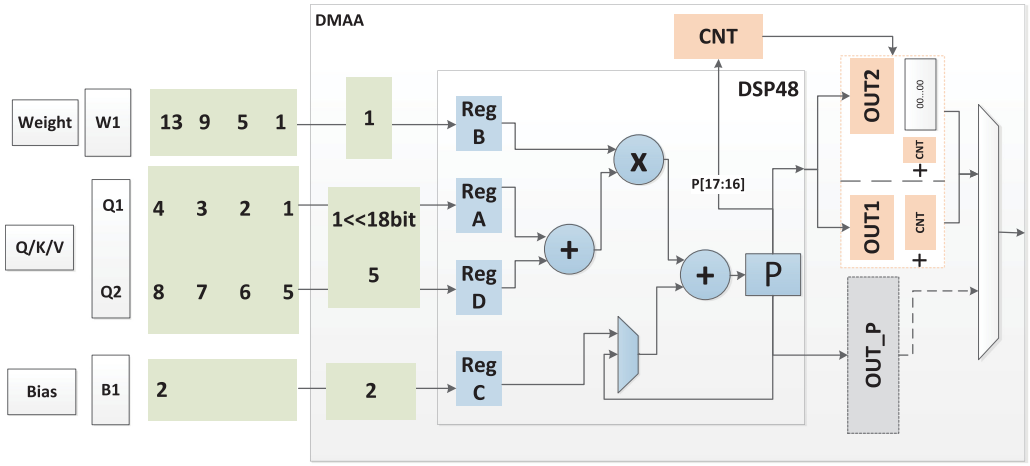
Compared with the traditional SA Architecture [21], another major feature of this paper is that the DMAC method [12] is used and improved, achieving twice the computational performance with the same resources. In addition, compared with the systolic array [25] built with logic resources, this paper uses DSP48 slice resources to complete the multiplication calculation with faster speed and low delay. The resources in the project will be well-distributed in this way, especially when they are tight. The compiler is easier to place and route with fewer timing issues and can ensure that FPGA runs at a relatively high clock frequency, which means it can process more data per unit clock.

As mentioned in subsection 4.2, SA is configurable because the core of FPE (DMAA and SMAA cores) can support mutual transformation. The architecture of DMAA and SMAA cores is depicted in Figure 8.

The difference between the two cores lies in input and output data processing methods. The implementation methods of the two cores are described below.

When the core is configured into DMAA mode, it can process the matrix multiplication of two rows of data and one column of data. As is shown in Figure 8(a), it can complete two 8bit multiplications, two accumulations, and two additions, $Acc(Q1 * W1) + B1$ and $Acc(Q2 * W1) + B1$. Ports A and D of the DMAA core can receive two parallel rows of matrix data, with a maximum of 8bit. One row of data is extended by 18 bits to port A of DSP48. The other row of data is inputted to port D. At the same time, another multiplied matrix data is inputted to port B, and the data to be added at the end is inputted to port C. DSP48 completes the multiplication and accumulation of data, the valid data of port C is added when it comes. Meanwhile, the overflow of low 18bit data is counted in the accumulation process (CNT). The result is divided into two parts, and adds the

(a) DMAA core



(b) SMAA core

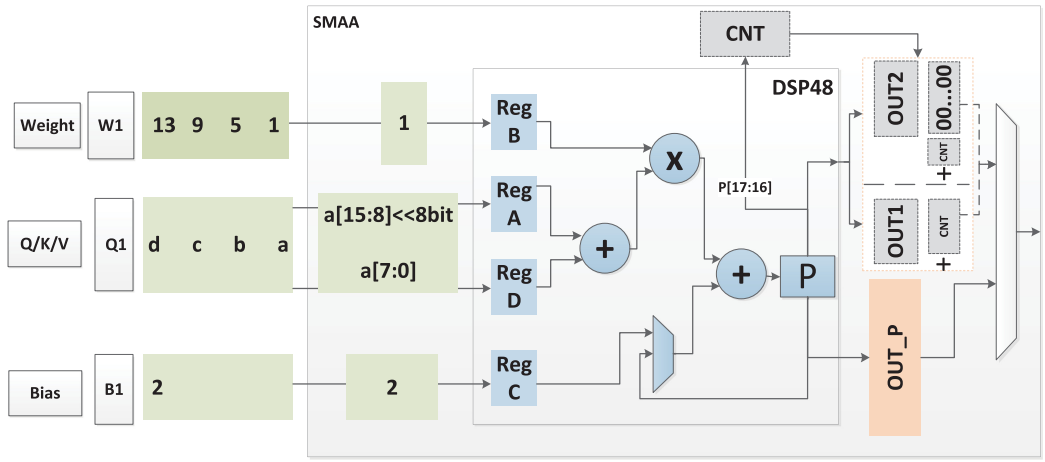


Fig. 8. The architecture of DMAA and SMAA cores.

counter to get the final results respectively, and then selects the output result according to the operation mode.

Compared to [12], it has the following advantages:

- No need for the guard bit;
- The shared operand can be signed;
- Using port C of DSP48, when the multiplication and accumulation are completed, the addition of data is also finished in the same DSP48 (this improvement can save the addition tree, compared with the literature[32]).

When the core is configured into SMAA mode, it can process the matrix multiplication of one row and one column of the input data. It can complete a 16bit multiplication, an accumulation, an addition, $Acc(Q1 * W1) + B1$. The row of matrix data is inputted into ports A and D of the SMAA core, and the maximum data is 16bit. The high 8 bits of data are extended to 16 bits by left shift and sent to port D of DSP48, and the low 8 bits of data are inputted into port A. Meanwhile, another

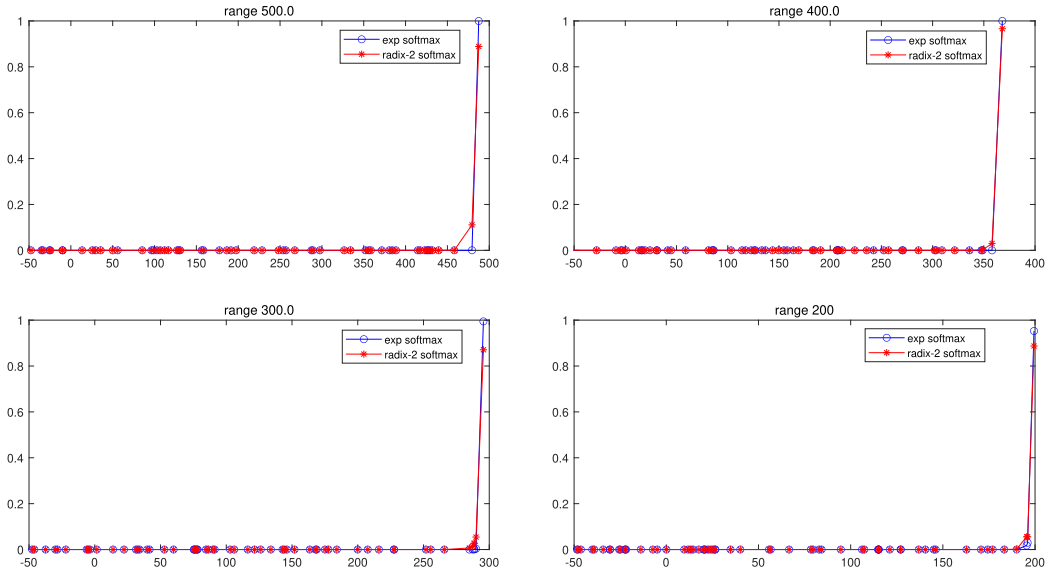


Fig. 9. The comparison of exp softmax and radix-2 softmax.

multiplied matrix data is inputted into port B. The data added at the end is inputted into port C. Finally, the output result is selected according to the operation mode.

4.4 Radix-2 Softmax

This paper proposes a new algorithm based on 2^x softmax (called radix-2 softmax) to make the softmax calculation more hardware-friendly. The traditional function of softmax(exp softmax) is as follows:

$$f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}} (i = 1, 2, \ldots N) \tag{3}$$

The radix-2 softmax scheme is proposed as follows:

$$f(x_i) = \frac{2^{x_i}}{\sum_{j=1}^{N} 2^{x_j}} (i = 1, 2, \ldots N) \tag{4}$$

The comparison of exp softmax and radix-2 softmax is shown in Figure 9. We simulate four groups of different range data, represented as range 500, range 400, range 300, and range 200. Each group has 100 samples, and the data samples are random. For instance, data in set range 200 are values in the range [−100, +100]. The data distribution patterns of radix-2 softmax are similar to exp softmax, and they can also complete the mapping function from discrete data to probability from 0 to 1. We have evaluated the mean error of radix-2 softmax compared to exp-softmax, and it is only 0.0022 in 20 million data samples.

The main calculations include two operations in radix-2 softmax, finding the highest bit and bitshift. It avoids the difficulty computation in softmax, the exponential calculation. This method can effectively save resources, improve computing speed and reduce delay.

The architecture is shown in Figure 10. The first part is to normalize the input data to 8bit, solved by bitshift after finding out the highest significant bit of the input data. The second part is to give out the 2^x through a look-up table. The third part is to output the final results with a division. Therefore, the data input to this architecture can be pipelined and calculated at high speed.

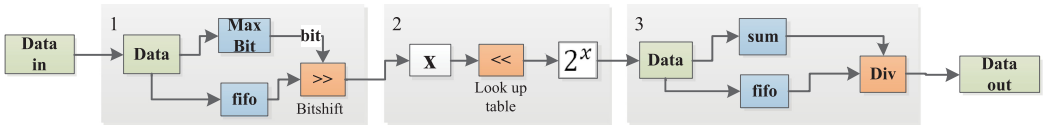


Fig. 10. The architecture of radix-2 softmax.

4.5 Bram Bw-Converter Cache Group and Bram Result Buffer group

The data is taken from DDR and sent to the **Bram Bit width (Bw)** converter cache group. The Bram Bw-converter cache group uses DDR's large bandwidth characteristic, converting data from 128 bit to 8bit or 16bit. It consists of multiply Brams, and the number depends on the row of SA. These Bram cache groups generate the timing sequence of SA input. The embedding size determines the depth of Brams.

In the architecture, each multi-head attention kernel uses three Bram result buffer groups to alternately cache the calculation results of two SAs to achieve load balancing. The scheduler will fetch data from Bram buffer groups to SAs or DDR at sometimes.

4.6 The Scheduler

In the architecture designed in this paper, there are two levels of schedulers. The first scheduler is responsible for the operation process control of the top architecture and completes the scheduling of data and instruction transmission for the multi-head attention kernels. There are four main steps between Host and FPGA Platform with software and hardware co-design, shown in Figure 11.

Step 1: The OpenCL environment is initialized in Host. In this section, the Host detects the attached Xilinx device, loads the FPGA binary file, and programs it into the Xilinx device. Then a command queue and the kernel object are created.

Step 2: The application creates the buffers needed to share data with the kernel and then maps the allocated memory to user-space pointers.

Step 3: The host program sets the arguments of the kernel, then schedules three operations: the transfers of the input vectors to device memory by the Axi4 data interface, the execution of the kernel by the Axi4-Lite control interface with AP_strart, and lastly, the transfer of the results back to host memory by the Axi4 data interface. These operations are enqueued in the command queue declared in Step 1.

Step 4: The Host calls to finish when FPGA gives AP_done after all previously enqueued operations are completed. In this case, it implies that the output buffer containing the kernel results has been migrated back to host memory and can safely be used by the software application.

The second scheduler is responsible for the operation process of the kernel, which completes the allocation and scheduling of SAs calculation. The core of the second scheduler is two levels of the state machine. The first level state machine includes a variety of scheduling processes that can be prefabricated to adapt to different calculation tasks, such as different sequence lengths or embedding sizes. The second level state machine controls the SAs and gives the parameters required for each calculation step, including the address and length of data reading and writing, SA mode, and the selection of the Bram cache and Bram buffer groups. Thus, the schedulers described above ensure this paper's architecture is reconfigurable and efficient.

In the design of FPGA architecture, considering the requirements of multiple calculation modes, each computing step is loosely coupled, and the input and output of SA modules are buffered and decoupled by Bram buffer groups. For example, for different d_{model} or sequence lengths, it is only necessary to select the first scheduler's corresponding secondary scheduler to adapt and quickly complete the required calculation configuration. Therefore, the architecture of this paper is flexible and can adapt to a variety of computing scenarios.

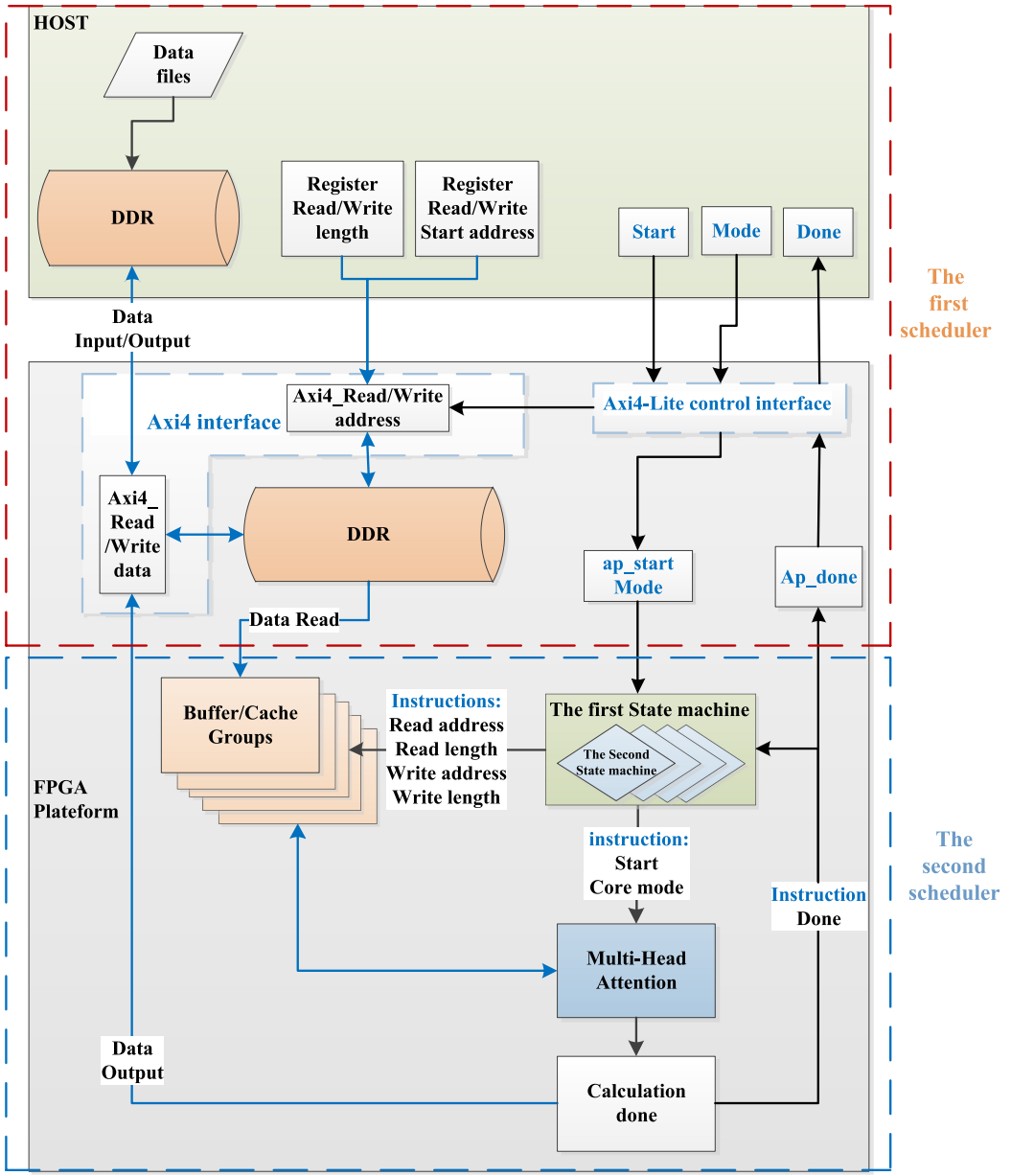


Fig. 11. The schedulers of the whole architecture.

5 EXPERIMENTAL RESULTS

5.1 Experimental Settings

5.1.1 Base Model. Our experiment is based on three Transformer models: a Transformer model (model 1) for a machine translation task [27], a shallow Transformer language modeling task (model 2) with the WikiText-2 dataset [22], and a BERT model (model 3) for sentence (and sentence-pair) classification tasks with GLUE data sets [31].

The machine translation Transformer has been trained and tested with IWSLT 2014 English to German parallel corpus. The test BLEU score is 27.3 with Transformer(base model) [27], containing 165k training tokens with a vocabulary size of 45k for German and 31k for English.

The language modeling Transformer is implemented on the WikiText-2 dataset, which contains 2M training tokens with a vocabulary size of 33k. This shallow Transformer model with four attention heads and 512 dimensions is established.

To reflect the practicality and adaptability of this architecture, we also carry out the experiment with a pre-trained BERT model [28] and fine-tune eight tasks from GLUE data sets.

For the small model, this paper takes $n = 32$ and 64, $d = 512$. Here n represents the maximum number of input sequences for the attention, and d represents the max position embedding dimension. To adapt to the transformer model with larger sentence length, such as the BERT, this paper takes $n = 128$, $d = 512$ to support the BERT-Base with GLUE data sets. There are different n and d in these models, Ftrans [14] deals with multiple designs, but the architecture of this paper is reconfigurable and we reuse the SAs with multiple second schedulers discussed in Section 4.6. So, the accelerator can be designed for any n or d.

5.1.2 Quantization. There are five steps in our acceleration calculation that need to design quantization carefully.

The linear transformations of the input sequence in step 1 and 2, which is shown in Figure 5 and Equation (2), quantize the matrix of Weight(W_q, W_k, W_v), Bias(B_q, B_k, B_v) to 16bit, and input sequence X to 8bit. We use SAs with DMAA core to calculate the results(Q_i, K_i, and V_i) and output in 16bit.

We implement the calculation of $Q * K^T$ in step 3 in SAs with SMAA core, where we quantize both of Q and K matrix in 16bit. The softmax result quantized to 16bit. Thus, we use SAs with SMAA core in step 4 to accelerate the calculation of Equation (1).

In the linear transformation of attention at step 6, we employ SAs with SMAA core in multi-head to calculate and quantize the matrixes to 16bit.

We evaluate the impact($\leq$0.22% and $\leq$1.54%) of the above quantization in the calculation of SA and attention when compared to the FP32 baseline with the model 1 discussed above.

5.1.3 Hardware Platform. The Xilinx Alveo U250 board, which has 2,688 of 36k BRAM, 12,288 DSPs, and 1,728k logic cells (LUT), is used in the experiment. The FPGA card is connected with the host machine through PCIe x16 for fetching the input words and weight matrixes. Xilinx Vitis 2020.2 is used for hardware development. The hardware inference performance speedup and energy saving are compared between Intel(R) Xeon(R) Gold 5220R CPU(2.20GHz) CPU, Nvidia Tesla P100 GPU, and Xilinx Alveo U250 FPGA platforms.

5.2 Performance Evaluation

5.2.1 Evaluation Methodology. In this paper, three Transformer applications are evaluated and analyzed. The experiments are implemented using Python 3.7.10, PyTorch 1.8.1, and CUDA 10.1.

We run attention with PyTorch and use cuDNN on GPU and CPU, torch.cuda.Event on GPU and time.clock on CPU are used to measure the latencies.

We get cycle-accurate latency with some logic resources in FPGA, also we use time.clock in Host C++ software to measure the total latencies. For power estimation, we get the power report from Vivado (a design tool of Xilinx Vitis 2020.2).

For latency measurements, we repeat 500 times, remove the largest 10% and smallest 10%, and average the remaining latencies.

This paper mainly uses the dynamic reconfigurable systolic array to accelerate the multi-head attention, which is the largest amount of computation in NLP. Firstly, extracting the parameters

Table 2. The Resource Utility of FPGA

Modules	DSP	BRAM	LUT
top	4	104.5	79k
Memory subsystem	9	636.5	129k
Attention kernel*4	1044*4	260*4	132k*4
Total	4189	1781	736k

in the trained PyTorch model and the embedded data after positional encoding, and then storing in multiple files after quantifying to int8 and int16. Therefore, before running the accelerated calculation, it is initiated by the Host to read the parameter files into the Host's memory and then input them into the DDR of the FPGA card through the PCIe X16 interface. The calculation part of the attention is carried out in the FPGA. The FPGA kernels will take the embedded data and weight matrixes from the DDR according to the reading address and length given by the scheduler, and store the calculated results in the DDR of the FPGA platform. At the same time, the scheduler will notify the Host to extract the calculation results and send them back to the Host's memory through PCIe. The schedulers of architecture are shown in Figure 11.

The FPGA project includes two parts. One part is that the program is written in Verilog language, and the bitstream file is generated after Vivado synthesis, place, and route. Here, we restrict the running frequency of the main program to 300MHz in the Vivado project and use the performance strategy for place and route. The other part is the host executable program written in c + + language and compiled through Vitis, in which we instantiate four accelerated kernels. The FPGA utilization report is presented in Table 2.

5.2.2 Latency. To verify the performance of this architecture, we firstly compare the most important characteristic (attention computing delay) with GPU and CPU. As shown in Figure 12, we implement three different transformer models with sequence lengths 32, 64, and 128, respectively. We achieve 51.3×, 24.5×, 31.2× speed up on FPGA compared to CPU and 17.3×, 6.2×, 3.3× speed up compared to GPU. We note that the amount of computation increases with the increase of sequence length, and the accelerating process changes from computing-dominated to memory-dominated. The acceleration ability of FPGA decreases mainly due to the following two reasons. First, the acceleration cores of the architecture in this paper are the SAs of (32, 32). When the sequence length exceeds 32, the internal scheduler will call the systolic array through multiple cycles. The corresponding embedded data, weight, and bias need to be read from DDR, and the result data should transfer to the Host many times. Therefore, the accelerated computing time ratio is relatively reduced. Second, the memory bandwidth in P100 GPU is almost ten times that of the Alveo U250 card, so GPU is faster in data scheduling. If we use the Alveo U280 card, the FPGA integrates the HBM memory, and the memory bandwidth reaches 460GB/s, which greatly improves the storage access speed and decreases the gap with the GPU.

It is deserved to mention that the author [27] implemented the Transformer himself in model 1, and the Transformer codes in model 2 are based on the NN Library of PyTorch. There is a certain performance gap between them, and the Transformer in NN Library is optimized on GPU. However, the FPGA developed under the architecture of this paper has achieved a certain amount of acceleration.

Because the architecture of this paper adopts an efficient systolic array and multi-head parallel calculation of attention, many pipeline processings are used in FPGA, which improves the performance of matrix multiplication calculation. Although the clock frequency of FPGA is lower than that of CPU and GPU, it can accelerate the calculation dozens of times.

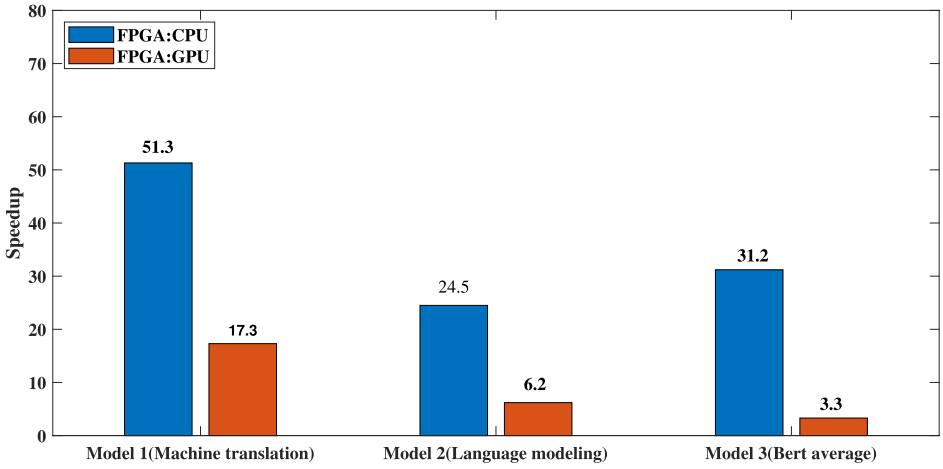


Fig. 12. The performance in different Transformer implementation.

We propose a comparison of attention latency with two papers about the Transformer accelerator using FPGA [21, 24], shown in Table 3. In the first experiment, the acceleration of 1.73× was completed when the sequence length was nearly twice that in literature [24]. In the second experiment, the acceleration of 1.33× is completed when the head number of this paper is half of the work [21]. It can be seen that the architecture of this paper achieves more computation and less latency.

5.2.3 Throughput. The hardware platform is composed of a high-performance server and Xilinx Alveo U250 card. The bandwidth of data exchange between server and FPGA through PCIe can reach an average of 11GB/s(host writes 9GB/s and host reads 12GB/s). Before the first acceleration, it is necessary to transfer the embedding data and weight matrix to the DDR of U250 card. This process will consume relatively more time. However, except for accelerating a new attention module, these data only need to be transmitted once. In the later operation process, FPGA only needs to get the corresponding data from DDR, which is much faster because the total bandwidth of DDR can reach up to 77GB/s. After the acceleration operation starts, the subsequent DDR data reading and writing process can be hidden in the calculation process by using pipeline technology to improve the throughput of the whole accelerator. The theoretical int8 peak throughput of the architecture in this paper is 4.9152TOP/s. According to statistics, the effective computing time occupies for 36.61%, so the throughput of the architecture in this paper is 1800GOP/s.

5.3 Energy Evaluation

5.3.1 Evaluation Methodology. We measure the power with nvidia-smi and hwinfo64 software for Tesla P100 GPU and Xeon CPU. We first record the system's idle power, and then repeatedly run workloads and get the total power and utility ratio. The dynamic power is total power minus idle power, and the energy is power multiply latency.

The power consumption of FPGA comes from the power report of the Vivado project. We set the power consumption evaluation parameters as follows:

(1) Airflow is 500lfm;
(2) Heat sink is high;
(3) Board selection is large;
(4) Other parameters are default.

Table 3. The Comparison of FPGA Implementation

Serial number	Accelerator	d_{model}	Head	Sequence length	Improvement
1	Peng et al. [24]	800	4	18	1.73×
	This paper	512	4	32	
2	Lu et al. [21]	512	8	64	1.33×
	This paper	512	4	64	

Table 4. The Comparison of Energy Consumption

Serial number	Accelerator	CPU		GPU		Energy Improvement	
		utilization ratio	Power(W)	utilization ratio	Power(W)	FPGA: CPU	FPGA: GPU
1	Machine translation Transformer	60.40%	76.47	83.47%	75.43	54.4×	17.9×
2	Language modeling Transformer	86.72%	99.04	99.89%	144.13	33.6×	11.2×
3	Bert (average)	88.76%	96.92	100%	153.52	41.9×	6.5×

5.3.2 Energy Consumption. To comprehensively measure the energy consumption, we give the utility ratio and CPU and GPU power consumption when the model is running. We calculate the energy consumption combined with the running time and give the energy saving rate of FPGA.

The power report of the FPGA shows that the total power consumption is 77.168w, the dynamic power consumption is 72.158w, and the junction temperature is 51.9°C. However, the Xilinx Alveo U250 FPGA card has a high power consumption because of its massive resources. It adopts Xilinx **Stacked Silicon Interconnect (SSI)** technology, containing four **Super Logic Regions (SLR)**. The card is equipped with an active cooling fan, so its junction temperature is controlled very well. In actual operation, the power consumption will be smaller than this evaluation because all the resources in FPGA are not running simultaneously.

The specific energy consumption data are shown in Table 4. Although the power consumption of FPGA is high, the energy saving is 54.4x, 33.6x, and 41.9x relative to CPU and 17.9x, 11.2x, and 6.5x relative to GPU.

5.4 Discussion

5.4.1 End-to-End Performance with FFN. Section 2.1 mentioned that the FFN module is mainly composed of two linear layers. Its calculation is mostly matrix multiplication, which is suitable for acceleration with systolic arrays. In addition, in the architecture of this paper, the calculation of the FFN layer can also be allocated to four acceleration kernels for parallel processing.

To compare the end-to-end performance with baselines. We extend our architecture to support the FFN layers by reusing the SAs and adding a new second scheduler. On BERT-Base benchmarks,

the FPGA-based FFN of this paper achieves, on average, 2.9× and 72.4× speedup over P100 GPU and Xeon CPU.

For evaluation of end-to-end LNP performance, we measure the latency of the whole BERT and outperform Ftrans [14] with 1.27x speedup in the batch size of 16.

5.4.2 Comparison with the other Accelerators. This paper compares with multiple attention accelerators in many aspects, as shown in Table 5. In terms of performance, our accelerator has reached the acceleration ratio of 8.14x, 1.12x, 1.27x, 3.4x, and 1.65x compared with A^3 [9], SpAtten [32], Ftrans [14], Sanger [20], and ELSA [10], respectively.

Here we focus on three recent works with good performance. Sanger proposed a hardware and software co-design framework that exploits the dynamic sparsity in attention via a reconfigurable architecture. They applied software to evaluate sparsity and reconfigurable hardware to accelerate attention. There are many interactions between software and hardware in the computing process, which will increase the system delay and cannot accurately evaluate the latency of the whole computing process with the simulation environment. ELSA devised a novel approximation scheme that significantly reduces the amount of computation by efficiently filtering out relations that are unlikely to affect the final output. ELSA needs the user to define a threshold P to determine the similarity. However, this threshold needs a lot of training data and many attempts to determine, and for different inference tasks, this threshold will greatly affect the final result. Ftrans compresses weights of Transformer-based models with a block circulant matrix-based representation. But the hardware accelerators adopt fixed architecture design, and there are multiple FPGA designs for different batch sizes.

One of the main differences among these accelerators is that this paper does not use any pruning algorithm. It mainly focuses on matrix multiplication, data (including input and weight data) caching, and process scheduling in attention, which is relatively closer to the underlying hardware applications. We achieve a considerable acceleration ratio and energy-saving rate by optimizing FPGA design architecture. This paper's architecture and the SA can be applied to other pruning algorithm accelerators and should reach a more powerful acceleration ability.

This paper uses an FPGA chip, and the compiled project files can be operated and tested on a Xilinx Alveo card. In contrast, A3, SpAtten, Sanger, and ELSA accelerators use the simulation method to evaluate their performance and power consumption. They have the following disadvantages: first, the time delay of data transmission between the chip and other hardware devices is not considered; second, the chip needs a long process from design to production, and the investment cost is very high. It is inferior to the FPGA scheme in terms of performance/cost ratio.

6 RELATED WORK

The Transformer model was first proposed in 2017 [29] for NLP tasks and has achieved rapid progress in many research areas. The attention mechanism is the core of the Transformer. There were many works focused on accelerating with the algorithm [7, 18], architecture [11, 13], and hardware [10, 44].

The attention mechanism is used for translation [15, 29] and language Understanding [2, 5] in NLP. The attention mechanism has also been used in computer vision tasks recently [16, 35, 37, 42]. We improved part of the attention mechanism and designed a computation framework to accelerate it on FPGA. Our work can apply to many of these attention-based models.

The FPGA is widely used to accelerate neural network computations. For custom FPGA devices, CNN [23, 30] and GCN [6, 40, 41] accelerators are proposed to accelerate NN processing for its high parallelism and low latency. Some papers [14, 21, 24] have built FPGA or ASIC hardware

Table 5. The Comparison of Attention Accelerators

Attention accelerator	Pruning Technique	Computation saving	Computing modules	Target device	Evaluation method	Effective throughput
A3	Greedy iterative approximation	1.72x	Dot-product	ASIC (40nm)	Written and Simulate with Chisel; Compiled in Synopsys;	221GOP/s
SpAtten	Head pruning Token pruning	3.00x	Parallel Matrix mul	ASIC (55nm)	Written with SpinalHDL; Simulate with Verilator; synthesize with Cadence	1610GOP/s
FTRANS	Circulant matrix	1.33x	Element-wise Vec-mul	FPGA (16nm)	Written with C/C++; Synthesize with Xilinx HLS; Running on FPGA	Unknown
Sanger	Prediction + thresholding	5.90x	Reconfigurable Systolic array	ASIC (55nm)	Written and Simulate with Chisel; Compiled in Synopsys; Place and route with Cadence	529GOP/s
ELSA	comparing keys' similarities with a threshold	2.76x	Approximate Self-attention Computation	ASIC (40nm)	Written and Simulate with Chisel; Compile, Placed and route with Synopsys;	1088GOP/s
This paper	none	2x	Efficient reconfigurable Systolic array	FPGA (16nm)	Written with Verilog; Synthesize and Implement with Xilinx Vivado and Vitis; Running on FPGA	1800GOP/s

accelerators for Transformer. Most of these articles adopt HLS tools, and the efficiency is needed to be improved with the high utilization of resources in FPGA.

The systolic array is widely used in neural networks. The biggest advantage is its efficient parallel processing ability. There are many articles on systolic array improvement. Xu et al. [38] and Cho et al. [3] focus on solving the **processing element (PE)** utilization rate in the small-scale convolution or depthwise convolution. Wang et al. [33] presents AutoSA, an open-source compiler framework for generating high-performance systolic arrays on FPGA. At present, most of them focus on the acceleration of CNN [19, 39, 43] and RNN [1]. To the best of our knowledge, this is the first architecture that uses an efficient reconfigurable systolic array based on FPGA to accelerate the attention Mechanism in Transformer.

7 CONCLUSION AND FUTURE WORK

In this paper, we propose an efficient acceleration framework for the attention mechanism in the Transformer. Our architecture includes a top-level framework responsible for scheduling

calculation processes and large bandwidth data transmission through PCIe, a pair of systolic arrays with efficient and dynamically reconfigurable cores, and a hardware-friendly softmax implementation. It can be seen from the experiment that our architecture can complete more computation and higher acceleration with fewer resources. Therefore, a well-designed architecture and implementation with Verilog will be helpful to take full advantage of the resources on FPGA and build an efficient hardware accelerator.

The architecture designed in this paper mainly focuses on FPGA hardware acceleration without considering the sparsity of computing data, which will be considered in future research to save more computing resources and make the Transformer achieve higher acceleration on FPGA.

ACKNOWLEDGMENTS

The authors would like to thank anonymous reviewers for their comments.

REFERENCES

[1] Nagadastagiri Challapalle, Sahithi Rampalli, Makesh Chandran, Gurpreet Kalsi, Sreenivas Subramoney, John Sampson, and Vijaykrishnan Narayanan. 2020. PSB-RNN: A processing-in-memory systolic array architecture using block circulant matrices for recurrent neural networks. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 180–185.

[2] Yixin Chen, Weiyi Lu, Alejandro Mottini, Li Erran Li, Jasha Droppo, Zheng Du, and Belinda Zeng. 2021. Top-down attention in end-to-end spoken language understanding. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 6199–6203.

[3] Hyungmin Cho. 2021. RiSA: A reinforced systolic array for depthwise convolutions and embedded tensor reshaping. *ACM Transactions on Embedded Computing Systems (TECS)* 20, 5s (2021), 1–20.

[4] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. 2019. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860* (2019).

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[6] Luchang Ding, Zhize Huang, and Gengsheng Chen. 2019. An FPGA implementation of GCN with sparse adjacency matrix. In *2019 IEEE 13th International Conference on ASIC (ASICON)*. IEEE, 1–4.

[7] Jiangjin Gao and Tao Yang. 2022. Face detection algorithm based on improved TinyYOLOv3 and attention mechanism. *Computer Communications* 181 (2022), 329–337.

[8] Kaiyuan Guo, Shulin Zeng, Jincheng Yu, Yu Wang, and Huazhong Yang. 2019. [DL] A survey of FPGA-based neural network inference accelerators. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* 12, 1 (2019), 1–26.

[9] Tae Jun Ham, Sung Jun Jung, Seonghak Kim, Young H. Oh, Yeonhong Park, Yoonho Song, Jung-Hun Park, Sanghee Lee, Kyoung Park, Jae W. Lee, et al. 2020. Aˆ 3: Accelerating attention mechanisms in neural networks with approximation. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 328–341.

[10] Tae Jun Ham, Yejin Lee, Seong Hoon Seo, Soosung Kim, Hyunji Choi, Sung Jun Jung, and Jae W. Lee. 2021. ELSA: Hardware-software co-design for efficient, lightweight self-attention mechanism in neural networks. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 692–705.

[11] Wenjia He, Yu Wang, Lizhen Cui, Ran Su, and Leyi Wei. 2021. Learning embedding features based on multisense-scaled attention architecture to improve the predictive performance of anticancer peptides. *Bioinformatics* 37, 24 (2021), 4684–4693.

[12] Sugil Lee, Daewoo Kim, Dong Nguyen, and Jongeun Lee. 2018. Double MAC on a DSP: Boosting the performance of convolutional neural networks on FPGAs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 38, 5 (2018), 888–897.

[13] Bin Li and Yuqing He. 2021. An attention mechanism oriented hybrid CNN-RNN deep learning architecture of container terminal liner handling conditions prediction. *Computational Intelligence and Neuroscience* (2021).

[14] Bingbing Li, Santosh Pandey, Haowen Fang, Yanjun Lyv, Ji Li, Jieyang Chen, Mimi Xie, Lipeng Wan, Hang Liu, and Caiwen Ding. 2020. Ftrans: Energy-efficient acceleration of transformers using FPGA. In *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*. 175–180.

[15] Xintong Li, Lemao Liu, Zhaopeng Tu, Guanlin Li, Shuming Shi, and Max Q-H Meng. 2021. Attending from foresight: A novel attention mechanism for neural machine translation. *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 29 (2021), 2606–2616.

[16] Youling Li. 2020. A calibration method of computer vision system based on dual attention mechanism. *Image and Vision Computing* 103 (2020), 104039.
[17] Jimmy Lin, Rodrigo Nogueira, and Andrew Yates. 2021. Pretrained transformers for text ranking: BERT and beyond. *Synthesis Lectures on Human Language Technologies* 14, 4 (2021), 1–325.
[18] Dongyang Liu, Junping Zhang, Yinhu Wu, and Ye Zhang. 2021. A shadow detection algorithm based on multiscale spatial attention mechanism for aerial remote sensing images. *IEEE Geoscience and Remote Sensing Letters* (2021).
[19] Zhi-Gang Liu, Paul N. Whatmough, and Matthew Mattina. 2020. Systolic tensor array: An efficient structured-sparse GEMM accelerator for mobile CNN inference. *IEEE Computer Architecture Letters* 19, 1 (2020), 34–37.
[20] Liqiang Lu, Yicheng Jin, Hangrui Bi, Zizhang Luo, Peng Li, Tao Wang, and Yun Liang. 2021. Sanger: A co-design framework for enabling sparse attention using reconfigurable architecture. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*. 977–991.
[21] Siyuan Lu, Meiqi Wang, Shuang Liang, Jun Lin, and Zhongfeng Wang. 2020. Hardware accelerator for multi-head attention and position-wise feed-forward in the transformer. *arXiv preprint arXiv:2009.08605* (2020).
[22] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2017. Pointer sentinel mixture models.(2017). arXiv. *arXiv preprint arXiv:1609.07843* (2017).
[23] Duy Thanh Nguyen, Tuan Nghia Nguyen, Hyun Kim, and Hyuk-Jae Lee. 2019. A high-throughput and power-efficient FPGA implementation of YOLO CNN for object detection. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 27, 8 (2019), 1861–1873.
[24] Hongwu Peng, Shaoyi Huang, Tong Geng, Ang Li, Weiwen Jiang, Hang Liu, Shusen Wang, and Caiwen Ding. 2021. Accelerating transformer-based deep learning models on FPGAs using column balanced block pruning. In *2021 22nd International Symposium on Quality Electronic Design (ISQED)*. IEEE, 142–148.
[25] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI Blog* 1, 8 (2019), 9.
[26] Aurko Roy, Mohammad Saffar, Ashish Vaswani, and David Grangier. 2021. Efficient content-based sparse attention with routing transformers. *Transactions of the Association for Computational Linguistics* 9 (2021), 53–68.
[27] Alexander M. Rush. 2018. The annotated transformer. In *Proceedings of Workshop for NLP Open Source Software (NLP-OSS)*. 52–60.
[28] Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Well-read students learn better: On the importance of pre-training compact models. *arXiv preprint arXiv:1908.08962v2* (2019).
[29] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*. 5998–6008.
[30] Shreyas K. Venkataramanaiah, Han-Sok Suh, Shihui Yin, Eriko Nurvitadhi, Aravind Dasu, Yu Cao, and Jae-sun Seo. 2020. FPGA-based low-batch training accelerator for modern CNNs featuring high bandwidth memory. In *Proceedings of the 39th International Conference on Computer-Aided Design*. 1–8.
[31] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461* (2018).
[32] Hanrui Wang, Zhekai Zhang, and Song Han. 2021. SpAtten: Efficient sparse attention architecture with cascade token and head pruning. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 97–110.
[33] Jie Wang, Licheng Guo, and Jason Cong. 2021. AutoSA: A polyhedral compiler for high-performance systolic arrays on FPGA. In *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 93–104.
[34] Meiqi Wang, Siyuan Lu, Danyang Zhu, Jun Lin, and Zhongfeng Wang. 2018. A high-speed and low-complexity architecture for softmax function in deep learning. In *2018 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*. IEEE, 223–226.
[35] Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. 2018. Non-local neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 7794–7803.
[36] Xuechao Wei, Cody Hao Yu, Peng Zhang, Youxiang Chen, Yuxin Wang, Han Hu, Yun Liang, and Jason Cong. 2017. Automated systolic array architecture synthesis for high throughput CNN inference on FPGAs. In *Proceedings of the 54th Annual Design Automation Conference 2017*. 1–6.
[37] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. 2015. Show, attend and tell: Neural image caption generation with visual attention. In *International Conference on Machine Learning*. PMLR, 2048–2057.
[38] Rui Xu, Sheng Ma, Yaohua Wang, Xinhai Chen, and Yang Guo. 2021. Configurable multi-directional systolic array architecture for convolutional neural networks. *ACM Transactions on Architecture and Code Optimization (TACO)* 18, 4 (2021), 1–24.
[39] Rui Xu, Sheng Ma, Yaohua Wang, and Yang Guo. 2021. HeSA: Heterogeneous systolic array architecture for compact CNNs hardware accelerators. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 657–662.

[40] Hanqing Zeng and Viktor Prasanna. 2020. GraphACT: Accelerating GCN training on CPU-FPGA heterogeneous platforms. In *Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 255–265.
[41] Bingyi Zhang, Hanqing Zeng, and Viktor Prasanna. 2020. Accelerating large scale GCN inference on FPGA. In *2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 241–241.
[42] Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. 2019. Self-attention generative adversarial networks. In *International Conference on Machine Learning*. PMLR, 7354–7363.
[43] Jingyao Zhang, Huaxi Gu, Grace Li Zhang, Bing Li, and Ulf Schlichtmann. 2021. Hardware-software codesign of weight reshaping and systolic array multiplexing for efficient CNNs. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 667–672.
[44] Xinyi Zhang, Yawen Wu, Peipei Zhou, Xulong Tang, and Jingtong Hu. 2021. Algorithm-hardware co-design of attention mechanism on FPGA devices. *ACM Transactions on Embedded Computing Systems (TECS)* 20, 5s (2021), 1–24.

Received 31 December 2021; revised 19 May 2022; accepted 10 July 2022