Unified Accelerator for Attention and Convolution in Inference Based on FPGA

Tianyang Li[1,2], Fan Zhang[1,2], Xitian Fan[3], Jianliang Shen[1,2], Wei Guo[1,2], and Wei Cao[4]
[1]Information Engineering University, Zhengzhou, China
[2]National Digital Switching System Engineering and Technological R&D Center, Zhengzhou, China
[3]Shanghai HONGZHEN Information Science & Technology Corporation, Shanghai, China
[4]Institute for Big Data, Fudan University, Shanghai, China
Corresponding Author: Fan Zhang Email: 17034203@qq.com

2023 IEEE International Symposium on Circuits and Systems (ISCAS) | 978-1-6654-5109-3/23/$31.00 ©2023 IEEE | DOI: 10.1109/ISCAS46773.2023.10182145

***Abstract*—Many models combining Transformers with convolutional neural networks (CNNs) for computer vision tasks have achieved state-of-the-art results. However, due to the different computation patterns between attention and convolution, using a dedicated Transformer or CNN accelerator will inevitably reduce the computing efficiency of the other. To overcome this problem, we propose a unified architecture for attention and convolution on FPGA. We reduce runtime overhead by offloading part of self-attention computations offline before inference. Furthermore, we present a unified mapping method according to the computing characteristics of attention-based and convolution-based models. This accelerator implements multi-head attention in Transformer, independent ResNet-50 and hybrid blocks of attention and convolution in BoTNet-50 at 200MHz on Xilinx Virtex Ultrascale+ XCVU37P. Experimental results show that the solution is nearly 3.62 times more energy-efficient than the NVIDIA V100 GPU, and the computational efficiency is 11.86% and 28.29% higher than the state-of-the-art Transformer and ResNet-50 accelerators, respectively.**

***Index Terms*—attention, convolution, neural network, hardware accelerator, FPGA**

I. Introduction

In the past decade, convolutional neural networks (CNNs) such as VGG [1], ResNet [2], GoogleLeNet [3], and EfficientNet [4] have been a great success in computer vision (CV). However, the Transformer-based models introduced from natural language processing (NLP) [5] have recently stirred up a lot of interest in the CV community. Many works try to combine CNN architectures with attention mechanism, e.g., DETR [6], or completely replace convolution, such as Vision Transformer (ViT) [7]. We are witnessing more and more attention-based models or CNN-Transformer hybrid architecture being proposed [8, 9], achieving state-of-the-art results, and topping the list of CV tasks.

As the scale of Transformer models gradually increases, the computation becomes more and more intensive. Similar to CNN accelerators, some research works deploy attention-based Transformer models on FPGA [10-12]. However, due to different parallel computing strategies between convolution and attention, these attention-based accelerator architectures are not sufficiently applicable to CNN. As shown in Fig. 1, convolution uses an aggregate function on the localized receptive field to derive the output value according to the convolution kernel weights, which greatly reduces the number of weights while using a weight-sharing mechanism. In contrast, the attention mechanism uses a weighted average operation based on the context of the input features. In inference, the weights used by convolution are fixed parameters determined by the model, independent of the input, while the input features determine the weights of attention. Unlike the single-channel matrix multiplication in attention, the input feature map of convolution is usually multi-channels. Therefore, a unified architecture needs to be designed to improve inference speed and energy efficiency.

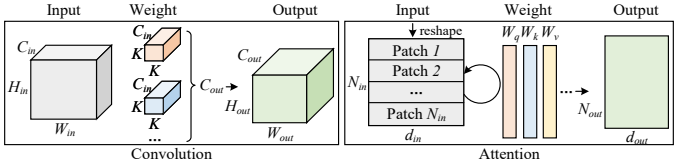


Fig. 1. Comparison with the computation of convolution and attention.

In this paper, we propose a unified architecture for attention and convolution on FPGA. It can efficiently compute self-attention and multi-head attention, as well as convolution, which are the most complex layers in Transformer and CNN. The contributions of this work are as follows:

- A computational optimization method for self-attention is proposed in inference. Offloading part of online computation to offline reduces the amount of computation on FPGA.
- A unified mapping method with weight sharing on FPGA is presented for convolution and attention. A well-designed computing flow ensures high hardware utilization of the computing kernel and on-chip memory.
- An FPGA-based unified accelerator is implemented for convolution and attention. Experimental results show that the computing efficiency of our proposed unified accelerator is 11.86% and 28.29% higher than the state-of-the-art Transformer and ResNet-50 accelerators, respectively. For the BoTNet-50 hybrid block, the computing efficiency achieves 71.24%.

The rest of this paper is organized as follows. Section II illustrates the computational optimization for self-attention and the unified mapping method for attention and convolution. Section III describes the proposed architecture in detail. Section

IV shows the experimental results. Section V concludes the paper.

II. Self-attention optimization and unified mapping method

A. Fast Self-Attention on FPGA

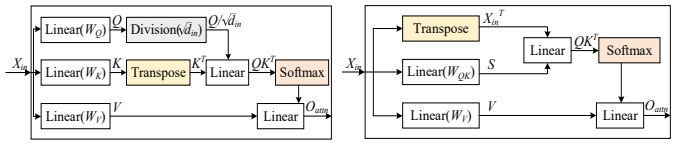
In standard self-attention, input X_{in} is multiplied by parameters W_Q, W_K, and W_V to obtain Query (Q), Key (K), and Value (V) for further computation, as shown in Fig. 2 (a). We try to reduce the amount of computation on FPGA by reducing the number of weight matrices. We note that:

$$\begin{aligned} QK &= \frac{Q}{\sqrt{d_{in}}} \cdot K^T = \frac{X_{in} \cdot W_Q}{\sqrt{d_{in}}} \cdot (X_{in} \cdot W_K)^T \\ &= X_{in} \cdot (\frac{W_Q \cdot {W_K}^T}{\sqrt{d_{in}}}) \cdot {X_{in}}^T \end{aligned} \tag{1}$$

We set $W_{QK} = \frac{W_Q \cdot {W_K}^T}{\sqrt{d_{in}}}$, Eq .1 can be rewritten as:

$$QK = \frac{Q}{\sqrt{d_{in}}} \cdot K^T = X_{in} \cdot W_{QK} \cdot {X_{in}}^T \tag{2}$$

Since W_Q, W_K, and d_{in} are constant matrices determined according to the model configuration in inference, W_{QK} can be computed offline before inference. FPGA only needs to load the weight matrix W_{QK} once to calculate QK, and the transpose of X_{in} replaces the transpose of K. The optimized self-attention on FPGA is shown in Fig. 2 (b), which we call fast self-attention. S is the intermediate result of X_{in} multiplied by W_{QK}. Compared with the standard self-attention, a Linear and a Division operations are reduced.



(a) Standard self-attention (b) Fast self-attention

Fig. 2. The computation flow of attention.

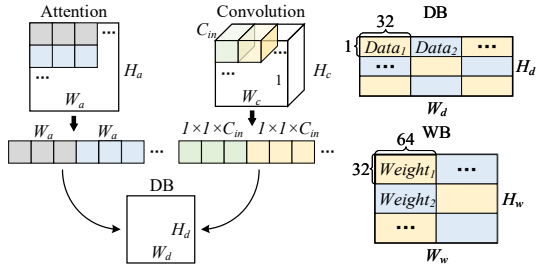
B. Unified mapping method for attention and convolution

To maximize the reuse of logic and memory resources, we map the matrix multiplication in convolution and attention simultaneously, which allows all general matrix multiplications (GEMMs) to be completed using the same computing kernel and on-chip memory. We tile the input feature map from attention and convolution into several same data blocks (DBs) with memory constraints and computing kernel parallelism, as shown in Fig. 3 (a). H_d and W_d represent the height and width of the DB. Weights are similarly mapped to some weight blocks (WBs).

To determine the scale of the FPGA computing kernel, we investigate typical Transformer and CNN models. Table I shows that the attention-based Transformers all have $d_{in} = 64h$, where h is the number of heads. In convolution-based CNNs, C_{in} is usually an integer multiple of 32 except for the first and last layers. At the same time, we consider weight sharing and reuse as many weights written on-chip as possible under a fixed multiplication and addition computation scale. Based on the above considerations, the DB and WB are partitioned separately, as shown in Fig. 3 (b). H_w and W_w represent the height and width of the WB. The computing kernel reads 1×32 data and 32×64 weights for computation in each clock cycle on FPGA, where each weight is reused H_d times. Based on this computing pattern, all of the GEMMs in attention and convolution can be done.

TABLE I
Examples of Transformer and CNN model configuration

	d_{in}	h	C_{in}
Transformer-base	512	8	-
ViT-B/16	768	12	-
Inception V3	-	-	3/32/64/80/192/288/768/1280/2048/1000
ResNet 50	-	-	3/64/128/256/512/1024/2048/1000



(a) Matrices tiling (b) Matrices partition

Fig. 3. Unified matrices mapping.

III. Hardware Architecture Design

A. Architecture Overview

The system architecture of the unified hardware accelerator for attention and convolution is shown in Fig. 4. The host parses the network description file, partitions the model, determines the scheduling sequence, and generates mapping instructions. Pre-compiled instructions, preprocessed input data (DB), and weights (WB) are stored in the off-chip memory (i.e., DRAM) before the runtime through the direct access memory (DMA) engine based on peripheral component interconnect express (PCIe). The global controller receives instructions to drive the unified accelerator, sends instructions to on-chip buffers, computing kernel, nonlinear and pooling unit, or sends DMA requests to off-chip memory.

The preprocessed data and weight are extracted from DRAM to the data buffer and weight buffer at runtime. The computing kernel reads data and weights from these buffers for computation. The output buffer is responsible for storing intermediate or final results. When the layer does not include nonlinear or pooling operations, the output of the computing kernel is directly written to the output buffer. Once the computation is complete, the final results are read from the output buffer and sent to the host.

B. Computing Kernel

We use the quantization method in [13] to replace FP32 with INT8 and achieve less than 1% accuracy loss on ImageNet2012. Therefore, the computing kernel processes DB and WB after INT8 quantization. As described in Section II.B, the computing kernel performs matrix multiplication of 1×32 data and 32×64 weights in each clock cycle, i.e.,

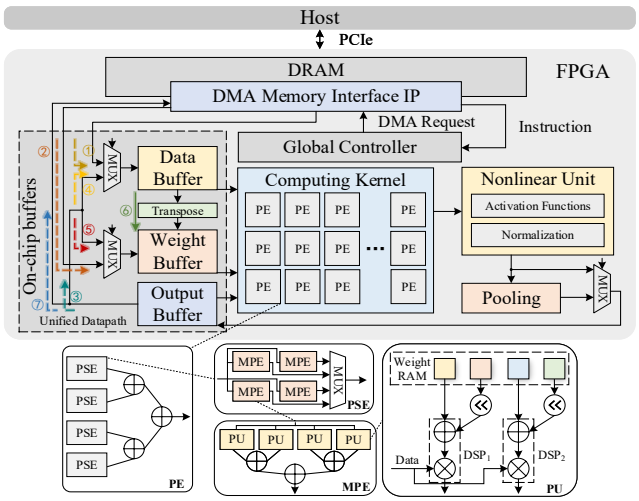


Fig. 4. System architecture.

2048MACs. The computing kernel is shown in Fig. 4, which consists of several process elements (PEs). Each PE contains 4 process subelements (PSEs), where PSE consists of 4 micro process elements (MPEs). An MPE is formed by connecting four process units (PUs) with an adder tree. Each PU uses 2 DSPs. For weight sharing in computation, we cache weights in the weight RAM using 4 LUTRAMs. When the weights are reused H_d times, it will be updated. The DSP48E2 in Xilinx FPGA can perform two parallel INT8 multiplication operations [14]. Using this method, we can reduce the usage of DSP slices in the computing kernel by half to 1024.

C. Unified Datapath

To adapt the accelerator to both attention and convolution, we design a unified datapath. As shown in Fig. 4, there are two channels for writing the data buffer from DRAM or output buffer (①, ④). For convolution, only one channel is used to write the weight buffer from DRAM (②). Due to the presence of intermediate weights and transposition in the attention computation flow, an additional output feedback channel ⑤ is used to write the intermediate weights (i.e., V) back to the weight buffer. Channel ⑥ is used to write the transposed results (K^T) to the weight buffer. Channel ③ is responsible for distributing the written back intermediate results to different input buffers, and the computation results are written to DRAM through channel ⑦. These channels can be configured during the runtime based on instructions. A matrix transposition module involved in channel ⑥ is implemented using a 32×32 LUTRAM with the ping-pong structure. Taking standard self-attention as an example, the computation flow using this unified datapath is shown in Fig. 5. By partitioning the computation, parallelizing matrix transposition and multiplication, most of the time overhead of accessing memory is hidden.

IV. Experimental Results

In this section, we implement the whole system architecture proposed in Section III with Vitis HLS (v2021.1) on Xilinx Virtex Ultrascale+ XCVU37P FPGA. The host machine adopted in our experiments is a server configured with multiple Intel(R) Xeon(R) Gold 5218 processors.

A. Evaluations on self-attention

By setting $N_{in} = d_{in} = 64$ in self-attention, we first evaluate the performance of standard self-attention and fast self-attention in Section II.A. The experimental results are shown in Table II.

TABLE II
Comparison with standard and fast self-attention

	Standard self-attention	Fast self-attention
MOP	2.62	2.10
Latency(us)	4.89	4.09
Speed-up	1×	1.20×
GOPS	53.58	51.34
Computing efficiency	65.44%	62.59%

The GOPS metric is determined by $Total\ ops/Latency$. The computing efficiency is calculated by Eq. 3, where the N_{op} represents the number of multiplications and additions in each clock cycle of the computing kernel. For computing kernels using DSP, $N_{op} = 2 \times N_{dsp} \times factor$. In our design, the DSP can complete two MACs simultaneously, so the factor is set to 2.

$$Computing\ Efficiency = \frac{Average\ GOPS}{N_{op} \times Frequency} \tag{3}$$

The experimental results show that the speed of fast self-attention is about 20% faster than standard self-attention, and the computing efficiency both exceeds 60%.

B. Hardware Performance Analysis

We implement independent multi-head attention in Transformer-base and independent convolution-based ResNet-50 on the unified accelerator, respectively, and compare our design with GPU and some previous FPGA implementations. Meanwhile, we evaluate the attention and convolution hybrid block in BoTNet-50 [15] on our accelerator.

An NVIDIA V100 GPU solution of multi-head attention in Transformer-base is reported in [10], and we ran the Tensorflow model of ResNet-50 on NVIDIA V100 GPU for comparison. Ref. [10] designed the first hardware accelerator for the multi-head attention in the Transformer. ShortcutFusion [16] optimized the FPGA accelerator for ResNet-50 by a reusability static memory allocation for shortcut data. BoTNet-50 introduces attention to ResNet-50, replacing 3×3 convolutions with multi-head attention in the last three bottleneck blocks of ResNet-50.

The experimental results are shown in Table III. For Transformer-base, we focus only on multi-head attention and use the number of multiplication and addition operations in multi-head attention to representing the model size. For ResNet-50, we directly utilize the model size to represent the total ops. Meanwhile, we evaluate the last three multi-head attention and convolution hybrid blocks in BoTNet-50. For the computing kernel implemented using LUT in [10], N_{op} is equal to the sum of the multipliers and adders. The energy efficiency is determined by $Throughput/Power$.

TABLE III
PERFORMANCE COMPARISON WITH GPU SOLUTIONS AND OTHER SOLUTIONS

	GPU [10]	**GPU**	**S. Lu [10]**	**ShortcutFusion [16]**	**Proposed**		
Network	Transformer-base*	ResNet-50	Transformer-base*	ResNet-50	Transformer-base*	ResNet-50	BoTNet-50*
Platform	NVIDIA V100		Xilinx XCVU13P	Xilinx XCKU115	Xilinx XCVU37P		
Process	12nm		16nm	20nm	16nm		
Frequency	1.46GHz		200MHz				
Data format	FP32		INT8				
Batch size	1						
Input size	64, 512	224×224, 3	64, 512	256×256, 3	64, 512	224×224, 3	64×64, 1024
GOP	0.11	7.74	0.11	11.76	0.11	7.74	23.09
DSPs	N/A		0	2240	1024		
Latency(ms)	1.5578	6.32	0.1067	11.69	0.1908	13.12	39.58
Throughput(frame/s)	641.93	158.19	9372.07	85.54	5241.09	76.22	25.27
Speed-up	1×		14.60×	0.82×	8.16×	0.48×	N/A
GOPS	70.61	1224.68	1030.93	1006	576.52	590	583.38
Computing efficiency	<1%	8.75%	62.92%	56.14%	**70.38%**	**72.02%**	**71.24%**
Power(W)	N/A	132	16.7	21.09	16.9	17.6	17.2
Energy efficiency(frame/J)	N/A	1.198	561.20	4.056	310.12	4.331	1.47

* The multi-head attention block in Transformer-base and the hybrid block in BoTNet-50 we adopt for experiments.

Experimental results show that the proposed unified accelerator achieves a performance of 576.52 GOPS on multi-head attention in Transformer-base, which is 8.16 times faster than GPU implementation. Compared with [10], the computing efficiency is improved by about 11.86%. For ResNet-50, the proposed unified accelerator achieves 590 GOPS, improves energy efficiency by 3.62 times compared with the GPU implementation, and improves computational efficiency by 28.29% compared with [16]. For the multi-head attention and convolution hybrid block in BoTNet-50, the unified accelerator achieves a performance of 583.38 GOPS.

C. Hardware Resource Analysis

The resource usages of the proposed unified accelerator and the comparisons with [10] and [16] are presented in Table IV. It consumes a total of 10.79% of the LUT, 8.56% of the FF, 11.11% of the BRAM, 6.67% of the URAM, and 13.96% of the DSP resources. All types of resources in our proposed accelerator are used more evenly than the designs in [10] and [16]. Therefore, the portability and scalability of our design are better than [10] and [16].

TABLE IV
RESOURCE USAGE COMPARISON WITH [10] AND [16]

Design	**Utlization**	**LUT**	**FF**	**BRAM**	**DSP**	**URAM**
Ours	Computing Kernel	106278	110229	0	1024	0
	Others	34425	112980	448	236	64
	Total	140703	223209	448	1260	64
	Avaliable	1303680	2607360	4032	9024	960
	Percentage	10.79%	8.56%	11.11%	13.96%	6.67%
S. Lu [10]	64×64 SA	420867	173110	0	0	0
	Others	50697	44749	498	129	0
	Total	471564	217859	498	129	0
	Avaliable	1728000	3456000	2688	12288	-
	Percentage	27.29%	6.30%	18.53%	1.05%	-
Shortcut Fusion [16]	Total	212.7K	361.5K	1184	2240	0
	Avaliable	663360	1326720	75.9MB	5520	-
	Percentage	32.06%	27.25%	55%	40.58%	-

V. CONCLUSION

We reduce runtime overhead by moving part of self-attention computations offline before inference. According to the computing characteristics of attention-based and convolution-based models, two types of large-scale computations are unified mapped, and the unified datapath and computation flow are well designed for the accelerator. Experimental results show that our design can efficiently compute attention and convolution independently and perform the hybrid computation of attention and convolution while outperforming previous solutions in terms of computing efficiency.

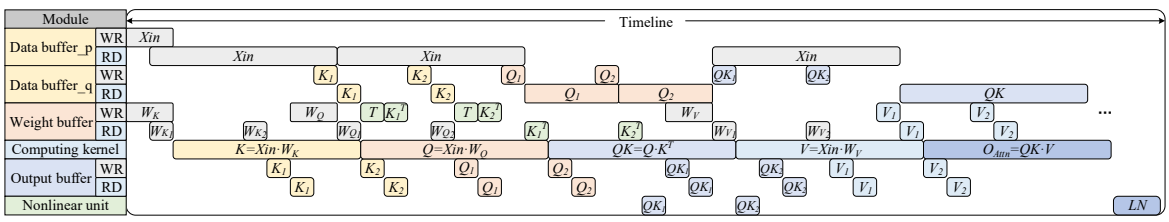


Fig. 5. The computation flow of self-attention on FPGA.

REFERENCES

[1] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-scale Image Recognition," 2014, arXiv:1409.1556.

[2] K. He, *et al.*, "Deep residual learning for image recognition," in *IEEE/CVF conf. on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770-778.

[3] C. Szegedy, *et al.*, "Going Deeper with Convolutions," in *IEEE/CVF conf. on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 1-9.

[4] M. Tan and Q. V. Le, "Efficientnet: Rethinking Model Scaling for Convolutional Neural Networks,"in *International Conference on Machine Learning (ICML)*, 2019, pp. 6105-6114.

[5] A. Vaswani, *et al.*, "Attention is All You Need," in *Advances in Neural Information Processing Systems (NIPS)*, 2017, 30.

[6] N. Carion, *et al.*, "End-to-end object detection with transformers," in *European Conference on Computer Vision (ECCV)*, 2020, pp. 213-229.

[7] Dosovitskiy A, Beyer L, Kolesnikov A, *et al.*, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale," 2020, arXiv:2010.11929.

[8] Z. Liu, *et al.*, "Swin Transformer: Hierarchical Vision Transformer Using Shifted Windows," in *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021, pp. 10012-10022.

[9] J. Zhang, *et al.*, "MiniViT: Compressing Vision Transformers with Weight Multiplexing," in *IEEE/CVF conf. on Computer Vision and Pattern Recognition (CVPR)*, 2022, pp. 12145-12154.

[10] S. Lu, M. Wang, S. Liang, J. Liu and Z. Wang, "Hardware Accelerator for Multi-Head Attention and Position-Wise Feed-Forward in the Transformer," in *IEEE 33rd International System-on-Chip Conference (SOCC)*, 2020, pp. 84-89.

[11] B. Li, *et al.*, "Ftrans: Energy-Efficient Acceleration of Transformers Using FPGA," in *ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED)*, 2020, pp. 175-180.

[12] X. Zhang, Y. Wu, P. Zhou, X. Tang and J. Hu, "Algorithm-hardware Co-design of Attention Mechanism on FPGA Devices," in *ACM Transactions on Embedded Computing Systems (TECS)*, vol.20, no. 5s, 2021, pp. 1-24.

[13] Y. Fu, *et al.*, "Deep Learning with INT8 Optimization on Xilinx Devices," Xilinx, Tech. Rep., 2017.

[14] Xilinx, "Deep Learning with INT8 Optimization on Xilinx Devices," in Xilinx WP486, 2017.

[15] A. Srinivas, *et al.*, "Bottleneck Transformers for Visual Recognition," in *IEEE/CVF conf. on Computer Vision and Pattern Recognition (CVPR)*, 2021, pp. 16519-16529.

[16] D. T. Nguyen, *et al.*, "ShortcutFusion: From Tensorflow to FPGA-Based Accelerator with Reuse-Aware Memory Allocation for Shortcut Data," 2021, arXiv:2106.08167.