ViA: A Novel Vision-Transformer Accelerator Based on FPGA

Teng Wang, Lei Gong, *Member, IEEE*, Chao Wang, *Senior Member, IEEE*, Yang Yang, Yingxue Gao, Xuehai Zhou, and Huaping Chen

***Abstract*—Since Google proposed Transformer in 2017, it has made significant natural language processing (NLP) development. However, the increasing cost is a large amount of calculation and parameters. Previous researchers designed and proposed some accelerator structures for transformer models in field-programmable gate array (FPGA) to deal with NLP tasks efficiently. Now, the development of Transformer has also affected computer vision (CV) and has rapidly surpassed convolution neural networks (CNNs) in various image tasks. And there are apparent differences between the image data used in CV and the sequence data in NLP. The details in the models contained with transformer units in these two fields are also different. The difference in terms of data brings about the problem of the locality. The difference in the model structure brings about the problem of path dependence, which is not noticed in the existing related accelerator design. Therefore, in this work, we propose the ViA, a novel vision transformer (ViT) accelerator architecture based on FPGA, to execute the transformer application efficiently and avoid the cost of these challenges. By analyzing the data structure in the ViT, we design an appropriate partition strategy to reduce the impact of data locality in the image and improve the efficiency of computation and memory access. Meanwhile, by observing the computing flow of the ViT, we use the half-layer mapping and throughput analysis to reduce the impact of path dependence caused by the shortcut mechanism and fully utilize hardware resources to execute the Transformer efficiently. Based on optimization strategies, we design two reuse processing engines with the internal stream, different from the previous overlap or stream design patterns. In the stage of the experiment, we implement the ViA architecture in Xilinx Alveo U50 FPGA and finally achieved ∼5.2 times improvement of energy efficiency compared with NVIDIA Tesla V100, and 4–10 times improvement of performance compared with related accelerators based on FPGA, that obtained nearly 309.6 GOP/s computing performance in the peek.**

Manuscript received 17 July 2022; accepted 26 July 2022. Date of current version 24 October 2022. This work was supported in part by the National Key Research and Development Program of China under Grant 2017YFA0700900 and Grant 2017YFA0700903; in part by the National Natural Science Foundation of China under Grant 62102383, Grant 61976200, and Grant 62172380; in part by the Jiangsu Provincial Natural Science Foundation under Grant BK20210123; in part by the Youth Innovation Promotion Association CAS under Grant Y2021121; and in part by the USTC Research Funds of the Double First-Class Initiative under Grant YD2150002005. This article was presented at the International Conference on Hardware/Software Codesign and System Synthesis (CODES + ISSS) 2022 and appeared as part of the ESWEEK-TCAD special issue. This article was recommended by Associate Editor A. K. Coskun. *(Corresponding authors: Lei Gong; Chao Wang.)*

The authors are with the School of Computer Science, University of Science and Technology of China, Hefei 230027, Anhui, China, and also with Suzhou Institute for Advanced Research, University of Science and Technology of China, Suzhou 215123, China (e-mail: wangt635@mail.ustc.edu.cn; greyyang@mail.ustc.edu.cn; gyingxue@mail.ustc.edu.cn; cswang@ustc.edu.cn; leigong0203@ustc.edu.cn; xhzhou@ustc.edu.cn; hpchen@ustc.edu.cn).

Digital Object Identifier 10.1109/TCAD.2022.3197489

***Index Terms*—Accelerator, field-programmable gate array (FPGA), vision transformer (ViT).**

I. Introduction

TRANSFORMER has been proposed by Google [1], and the research of natural language processing (NLP) has achieved significant development [2], [3], [4]. From 2017 to the present, models like GPT [5], BERT [6], and RoBERT [7] have been designed one by one and achieved significant improvements in NLP tasks, such as machine translation, language modeling, semantic analysis, etc. It has verified the ability of the attention mechanism in the transformer about the feature extraction and analysis of sequence data. In the past few years, the booming development of transformers in NLP has also affected the research of computer vision (CV) [8], [9], [10], [11], [12]. There is the calculation pattern used in the transformer differs from the pattern, which uses convolution units to extract the partial information in the convolution neural network (CNN) [1]. It focuses on the similarity in channels of a pixel with others pixels rather than the feature value in the receptive field. So far to now, Transformer-type models are rapidly surpassing CNN-type models in various image tasks [13]. As shown in Table I, the upper part is the development of the CNN model, and the lower part is the development of the vision transformer (ViT). When the transformer unit is used in the CV at first, the performance only achieves the score of ResNet [14] in 2015. However, the swin transformer [13] exceeds numerous CNN models and achieves 84.5% Top-1 accuracy in the ImageNet. It has verified the availability of the transformer's attention mechanism when used in CV.

However, the cost of the performance improvement is that the transformer-based models have millions of up to billions of parameters and a massive amount of calculation, like the GPT-2 used in WikiText-2 has 1542M parameters [5]. For these reasons, previous researchers have designed and proposed some accelerator structures based on field-programmable gate array (FPGA) for the transformer to deal with NLP tasks efficiently [15]. To reduce the amount of data and calculations, some previous works design different pruning strategies in transformer models and implement them in the hardware architecture [16], such as TEC [17], FT [18], and FTRANS [19]. Moreover, some previous works use different quantification mechanisms to reduce the overall computational

TABLE I
SUMMARY OF DIFFERENT BACKBONES ON IMAGENET-1K CLASSIFICATION. THE UPPER IS THE DEVELOPMENT OF THE CNN MODEL, AND THE LOWER IS OF VIT

Model	Time	Num. of Parameters	Num.of FLOPs	ImageNet top-1 acc.
AlexNet [23]	2012	60M	720M	57.2%
VGGNet [24]	2014	138M	15.3G	71.5%
GoogleNet [25]	2014	6.8M	1500M	69.8%
ResNet [14]	2015	55M	2300M	78.6%
DenseNet [26]	2017	25.6M	1150M	79.2%
SENet [27]	2017	145.8M	42.3G	82.7%
NASNet [28]	2018	88.9M	23.8G	82.7%
EffNet [29]	2019	30M	9.9G	83.3%
RegNetY [30]	2020	84M	16G	82.9%
ViT [8]	2020	307M	190.7G	76.5%
DeiT [31]	2020	86M	55.4G	83.1%
PVT [32]	2021	61.4M	9.8G	81.7%
Swin [13]	2021	88M	47G	84.5%

intensity of the model [20], [21], such as TEC [17], NPE [15], and FTRANS [19]. Meanwhile, ELSA [22] and NPE [15] focus on many nonlinear calculation units in the calculation process of the transformer model. These purposing approximate calculations replace these units to improve the overall calculation performance.

Moreover, there are apparent differences between the image data used in CV and the sequence data used in NLP. The details of models contained with transformer units in these two fields are also different. The difference is not noticed in the existing related accelerator design. Therefore, it is necessary to propose an accelerator architecture to execute the Transformer-type models in CV efficiently.

In terms of the data structure, the number of dimensions of image data used in CV is 4, while the expression form of text data used in NLP is the sequence, and the number of dimensions is 3. Furthermore, the ViT processes the image data to a series of patch tokens [8] and finds the relationship with patch tokens with multihead self-attention. So due to the calculation process of self-attention, the locality of image data is different from sequence data. The image data has more split dimensions and redundant information than sequence data. Therefore, the ViT potentially has a higher degree of parallelism in the data stream and higher performance when mapping this strategy to the computing units in the hardware platform for efficiency calculation.

In terms of the computing flow, because of the shortcut mechanism [14] in the transformer, the path dependence in the overall calculation affected the hardware calculation with the pipeline. To eliminate the impact, more on-chip resources are required for these additional data to storage [33]. However, the ViT only uses the encoder module in the whole model, which makes the distribution of the shortcut mechanism regular. Therefore, by using the regularity of the shortcut, modifying the module size, which maps to the hardware, can effectively eliminate the impact of path dependence and reduce the overhead of on-chip resources.

In this work, we propose the ViA, a novel ViT accelerator architecture based on FPGA, to execute the transformer application efficiently. By observing the computing flow of the ViT, we use the half-layer mapping and throughput analysis to reduce the impact of path dependence caused by the shortcut structure. At the same time, by analyzing the data flow in the ViT, we design an appropriate partition strategy to improve the efficiency of computation and memory access. Moreover, we develop an accelerator based on two reuse processing engines with the internal stream, obtaining nearly 309.6 GOP/s computing performance. Our main contributions are summarized as follows.

1) We conduct a thorough analysis of previous works, find the potential optimization opportunity of transformer hardware acceleration in CV, and propose a novel accelerator architecture for ViT, called ViA, which has two reuse processing engines with the internal stream.
2) For the shortcut mechanism in the calculation process of ViT, we use half-layer mapping and throughput analysis to reduce the impact of path dependence overhead and reduce the overhead of on-chip resources in theory.
3) By analyzing the input and output data characteristics of ViT, we design an appropriate partition strategy to solve the locality of image data, which improves the overall calculation and memory access efficiency.
4) Through the verification in the experimental part, the ViA we deployed on Xilinx Alveo U50 FPGA, which has a ~5.2 times improvement of energy efficiency compared with NVIDIA Tesla V100, and 4–10 times improvement of performance compared with related accelerators based on FPGA and achieved 309.6 GOP/s throughput performance.

The rest sections of this article are organized as follows. Sections II and III introduce the background of the transformer model and our motivation. Section IV describes the overall accelerator architecture proposed in this work. Section V mainly explains the methods we used in this work. Section VI mainly illustrates the results of the accelerator we implement and compares it with the previous work. The remaining sections complete the conclusion and related acknowledgments, respectively.

II. BACKGROUND

A. Vision Transformer

When the transformer was designed in NLP, it was considered that facing several 1-D sequence data. Each word is mapped into a feature vector through the embedding mechanism. Then, the results can be achieved when the embedded data through the encoder to the decoder layer by layer. Moreover, in CV, ViT modified the overall model structure, as shown in Fig. 1. First, researchers remove the decoder unit to simplify the model structure. Second, divide the image data into fixed-size patches to avoid the massive calculation of the similarity of all pixels' features [8]. As shown

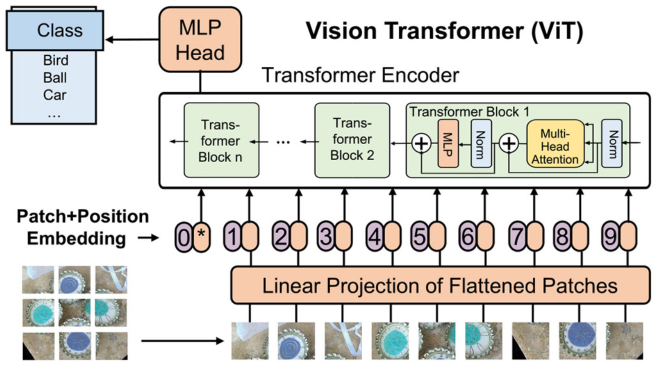

Fig. 1. Structure of the ViT model [8].

in Fig. 1, we can clearly understand the calculation process of the entire ViT. After the position embeddings, in the transformer encoder, data will be processed by *n* transformer blocks sequentially, containing layer normalization, multilayer perceptron (MLP), multiheads self-attention (MSA), GELU, etc. Finally, the results will be obtained after the output of the transformer encoder is calculated by MLP. Table II shows the primary calculation process of the transformer block, which is mainly divided into four parts for calculation, such as Layer Normalization 1, MSA, Layer Normalization 2, and MLP. Furthermore, there is potential parallel space in the overall calculation process, and the specific details are expanded in Section V-B.

Although ViT reduces the overall calculation amount by dividing the picture into a fixed size of patches as tokens, the complexity is still very high, proportional to the token's square. In order to solve these problems, Pyramid ViT (PVT) uses the pyramid structure from the CNN. It reduces the number of tokens layer by layer through multistage convolution operations [32]. Meanwhile, Swin Transformer not only uses the pyramid structure but also stitches the pixels in each <2, 2> area between each stage to further reduce the number of tokens through a linear mapping [13]. Moreover, Swin Transformer has added shifted window so that the boundary information between each patch can be retained between the network layers, thereby obtaining better image task performance than ViT.

B. Shortcut Mechanism

The key idea of the shortcut mechanism is residual modules in the middle of layers [14]. Since deep learning was proposed, the size of the model and the number of layers have been increasing. In theory, a deeper model can achieve better results. Moreover, utilizing standardized methods and normalized layers can also solve the gradient disappearance and gradient explosion problem in deep networks. However, Kaiming found that deep networks are also affected by network degradation, that deep networks do not perform as well as shallow networks under the same conditions. For this reason, Kaiming designed the residual module shown in Fig. 2 to solve the problem of network degradation.

This kind of thought greatly influenced the later development of deep neural networks. As shown on the paper [34], the

TABLE II
COMPUTATION FOR THE TRANSFORMER BLOCK IN VIT WITH N_{head} ATTENTION HEADS. X_0 REPRESENTS THE INPUT TO THIS UNIT. $X_1 - X_6$ REPRESENTS SUBSEQUENT OUTPUTS AFTER EACH OPERATION. $W_{\text{Mlp}i}$ REPRESENTS THE WEIGHT DATA FOR THE iTH Mlp LAYER

Layer Normalization 1
$X_1 = LayerNorm(X_0)$
Multi-Heads Self-Attention
$Q = X_1 \times W_Q; K = X_1 \times W_K; V = X_1 \times W_V;$ $Sim[0 : N_{head}] = Softmax(\frac{Q \times K^T}{Scale})$, Scale is a constant $Attn = Sim \times V$ $X_2 = Attn \times W_{proj}$ $X_3 = X_2 + X_0$
Layer Normalization 2
$X_4 = LayerNorm(X_3)$
Multi-Layer Perceptron
$Mlp = GELU(X_4 \times W_{Mlp0} + B_{Mlp0})$ $X_5 = Mlp \times W_{Mlp1} + B_{Mlp1}$ $X_6 = X_6 + X_3$

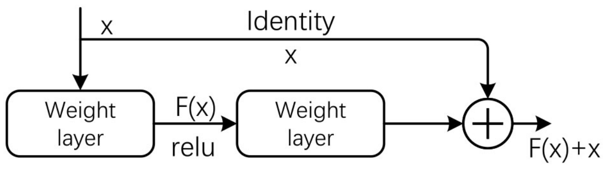

Fig. 2. Computing example of shortcut mechanism which is presented by Kaiming [14].

researcher indicates that the residual module can effectively reduce the attenuation of the gradient correlation caused by the back-propagation. However, this also brings unavoidable challenges to the design of the hardware accelerator pipeline structure. Due to the problem of the path dependence caused by the shortcut mechanism, the input data of some network layers needs to be reserved to avoid the bus access overhead caused by the merging process. This module may consume $P-1$ on-chip storage resources according to the input data size. P is the number of pipeline stages [33]. Therefore, designing the accelerator structure for such a network model requires further optimization. For details, please refer to Section V-A.

III. MOTIVATION

A. Difference of Transformer in CV

The previous introduction illustrates that the transformer is modified in the CV field. The essential modification is dividing the image data with patch size. The original intention of this modification is to reduce the vast computational overhead caused by global attention without accuracy loss. At the same time, it also implicitly increases the parallel dimension combined with the data characteristics of image data. As shown in (1), we use the critical calculation process in transformer

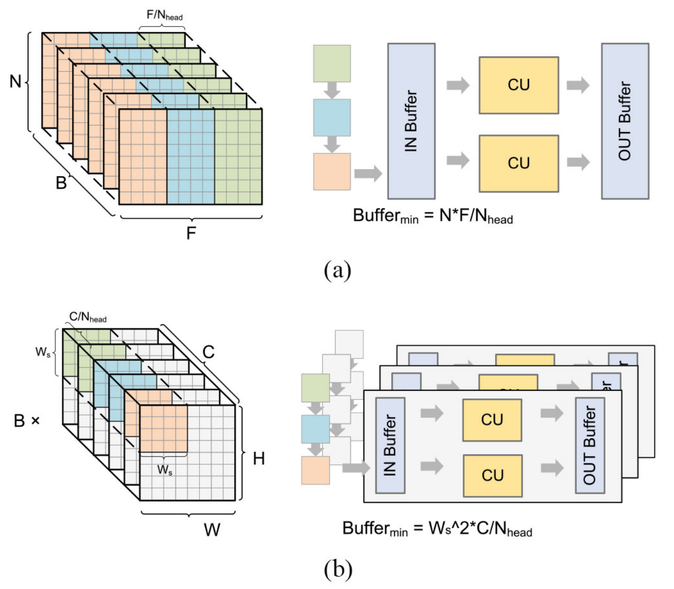

Fig. 3. Difference of independent data in the calculation. (a) Sequence data in NLP. (b) Image data in CV.

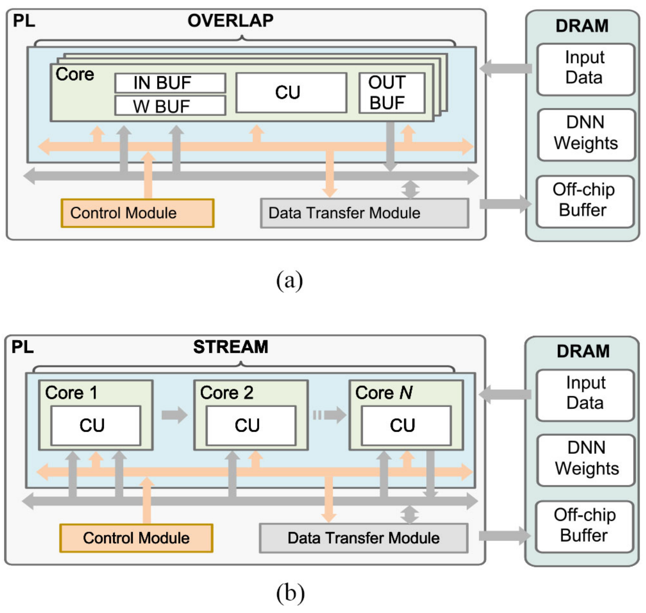

Fig. 4. Common accelerator architecture based on FPGA. (a) Structure of overlap pattern. (b) Structure of stream pattern.

unit MSA as an example to explain it

$$\begin{cases} \text{OUT}_{\text{attn}} = \frac{\exp(M_{QK})}{\vec{1}^T \times \exp(M_{QK})} \times V \\ M_{QK} = \frac{Q \times K^T}{\sqrt{\text{Scale}}}. \end{cases} \tag{1}$$

In the above formula, Q, K, and V are the query, key, and value matrix as the input of attention calculation, and M_{QK} is the intermediate result of the calculation performed by Q and K. Scale is a constant number, which is always the size of the feature, and $\vec{1}^T$ is a vector with one. The main dimensions of data used in CV are $<B, H, W, C>$, and the main dimensions of data used in NLP are $<B, N, F>$. The B is the batch size, H, W is the resolution of image data, C is the channel size, N is the sentence length, and F is the future size after embedding. Therefore, with the multiheads size N_{head} and patch size P, the independent data of calculation is represented as $Q, K, V \in \mathbb{R}^{H*W*C/(P*P*N_{\text{head}})}$ in CV, represented as $Q, K, V \in \mathbb{R}^{N*F/N_{\text{head}}}$ in NLP, as shown in Fig. 3.

W_S in Fig. 3 is the window size, where is H/P or W/P. Because of the $W_S \ll N$ and usually $C < F$, the independent data in CV is smaller than it in NLP. And it can be split by the dimension H and W, so the ViT can potentially achieve a higher degree of parallelism and map it to the hardware computing unit for efficiency calculation.

B. Design Patterns of Accelerator

In the implementation of the computing accelerator based on FPGA, the current design of hardware architectures is divided into two types [35], [36], [37], the overlap pattern and stream pattern. The thought of the overlap method is mainly to design a reusable big processing engine to perform the calculation part of the algorithm [38]. At the same time, the hardware control and operation scheduling are executed by the software in the upper layer [39], [40]. Therefore, it can be expanded according to the input model and the available FPGA resources, as shown at the upper of Fig. 4. The advantage of this design is flexible, which we do not need to be reconfigured when the input model has changed, but the computing type has not changed [41]. However, due to a control mechanism similar to the general processor, it is not easy to achieve maximum efficiency in computing [42]. Because this one-size-fits-all approach may lead to the inconsistent final performance on network models with different workload characteristics [43], [44], this also appears on ViT, and there are many kinds of nonlinear functions between each computation part. So if running the model in the overlap pattern, it may be challenging to achieve the optimal design size of the computing unit and the optimal schedule of the processing.

Another design pattern, stream, is opposite to the overlap. It implements different hardware units for each computing part in the calculation flow of the target algorithm and optimizes them separately to take advantage of interblock parallelism [45], [46]. As shown in the bottom of Fig. 4, the computing units are connected in the order of the algorithm. After the data flow through each unit, the calculation can be completed and flowed into the next unit [47]. Therefore, the advantage of this design pattern is that according to the difference of the characteristics in different computing parts, we can use different ways to optimize the implementation and parallel scheme in each hardware kernel [48]. However, the disadvantage is not flexible, and we should design new hardware computing units for different network models. ViT, which has multiple processing functions, is more suitable for stream patterns. However, the shortcut mechanism, appearing everywhere in the calculation, makes it difficult to improve performance efficiency in the accelerator. So we design an effective mapping strategy to reduce its harmful influence. For details, please refer to Section V-A.

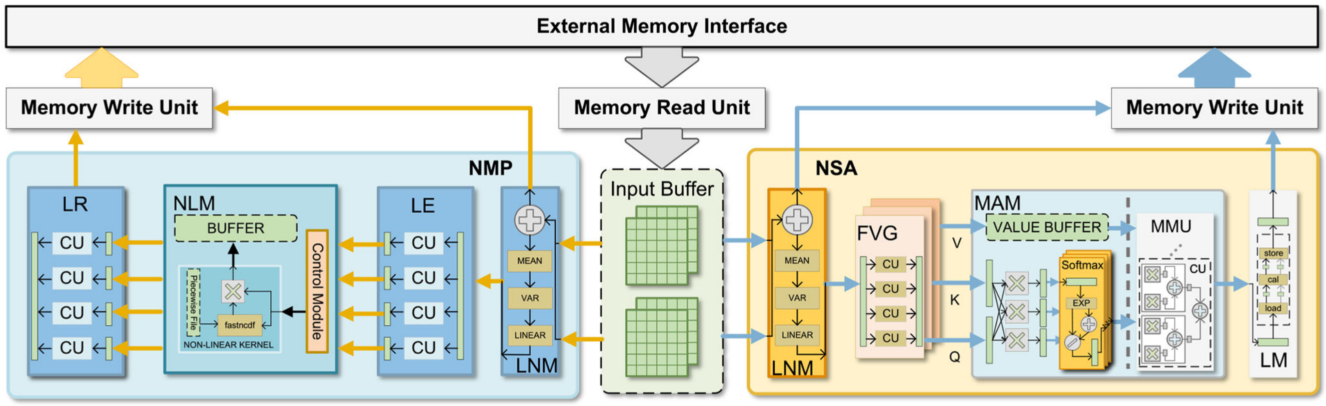

Fig. 5. Overall architecture of our ViT accelerator.

IV. ViA Accelerator Design

In this section, we present the overall architecture of ViA, the ViT accelerator. We make it out of multikernels with the stream structure. The details are as follows.

A. Overall Architecture

Fig. 5 presents the proposed architecture of our accelerator, which consists of the memory read unit (MRU), the memory write unit (MWU), the input buffer, and two types of processing elements (PEs), norm self-attention module (NSA) and norm MLP module (NMP). When the input data is loaded in the buffer by MRU, the controller dispatches the data to the NMP or NSA separately for efficient local computation. NSA is mainly responsible for calculating the layer-norm layer and multihead self-attention layer in the ViT model. At the same time, NMP is mainly responsible for calculating the layer-norm layer and MLP layer. Following the network structure of the ViT in Fig. 1, we can find the order of the processing transformer in the ViA. When calculating the result of the transformer encode, the ViA first uses the NSA unit to calculate the result of the input data-1 in the first half of the transformer block-1 and then sends it back to the external storage. The NMP unit continues to calculate the result in the second half of block-1. Meanwhile, the NSA units are also parallelized and calculate the result of input data-2 in the first half of transformer block-1. And so on, the calculation results of data-1 and data-2 are crossed and processed in the two PEs, NSA and NMP, respectively, and finally, the outputs of transformer blocks-2 to n can be obtained. The reason for this division is described in Section V-A. The logical connection between these components and PEs is shown in Fig. 5. Therefore, the performance of ViA is shown in

$$P_{\text{ViA}} = (P_{\text{NSA}} + P_{\text{NMP}}) * \#\text{ of Kernels}. \tag{2}$$

The performance of ViA is the number of kernels times the performance of NSA and NMP in theory. Meanwhile, because of the internal stream and parallel performance in PEs, we need to balance the workload of each part in the detailed design. Moreover, we introduce the design thought of these two PEs as follows.

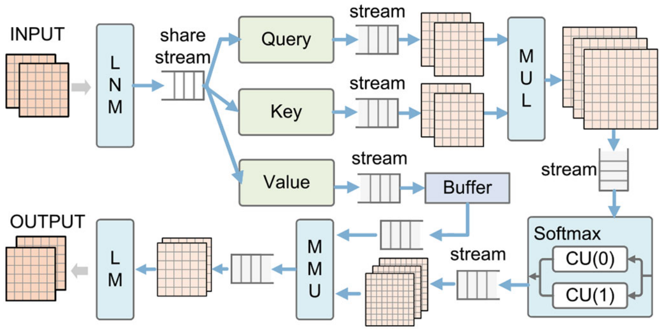

Fig. 6. Convert of input data through the calculation in NSA.

B. Norm Self-Attention Module

The structure of the NSA is shown in Fig. 5, which is composed of a layer norm module (LNM), a feature vector generator (FVG), a multihead attention module (MAM), and a linear-mapping module (LM). The input data is calculated by the LNM and becomes the standardized data. Then, the FVG generates parallel query, key, and value feature sets. Then, the data flows through the MAM to obtain this patch's characteristic similarity value matrix. Finally, we can obtain the output through the LM. The calculation complexity of the NSA is shown in

$$\Omega(\text{NSA}) = \left(2 + 2W_S^2\right) * H * W * C + 4 * H * W * C^2. \tag{3}$$

Each module has different computing units to achieve different functions, and the detail of the calculation flow is shown in Fig. 6. The LNM contains three operations: 1) loading data; 2) computing statistical parameters; and 3) standardized output. LNM loads part of the input data into the computing unit's local buffer, calculates the mean and variance of the data under the feature dimensions, and standardizes data to output. Therefore, the overall calculation time of LNM is probably within the level of $H * W * C * 4$. Moreover, the FVG needs to calculate the feature set of the multihead self-attention mechanism. Here, we broadcast the input data triple times to the computing unit to calculate the results of the query, key, and value set parallelly. Therefore, the overall calculation time of FVG is probably within the level of $R * C^2$. Then, the MAM is split into two parts, MAM-1 and MAM-2. MAM-1 calculates the feature similarity result of the query and key, while MAM2 calculates the product of the similarity matrix and the value to obtain the self-attention result. Furthermore, these calculation times for two parts are about $N*W^2*C+N*W^2$ and $N*W^2*C$. The processing is divided into two parts because the value set in the output result of FVG needs to be used in MAM-2. To improve the efficiency of the calculation pipeline, by caching the value for one stage, the performance of the MAM can be increased to double without additional computing resource overhead. Finally, the LM calculates input with linear mapping and outputs the result. The overall calculation time is probably within the range of $R * C^2$.

Therefore, to ensure efficient processing in this stream computing structure, it is necessary to ensure that the throughput of each stage is similar. We make the throughput of the LNM as a baseline. The parallelism required by the FVG should be $C/4$. The parallelism required by the MAM-1 should be $W * C/(4 * C - W)$, and MAM-2 should be $W/4$. Finally, the parallelism of the LM should be $C/4$. This way, it can ensure that we could achieve the overall computing performance with the full utilization of the computing power in this structure, as shown in

$$P_{\mathrm{NSA}} = \frac{\mathrm{Frequency} * \mathrm{FLOPs}}{\mathrm{Max}(\mathrm{Latency}_i)}, \quad i \in \mathrm{NSA}. \tag{4}$$

For example, assuming that the input dimension is <6, 6, 24>, the number of multiheads is 3, and the patch size is 3. After the input passes through the LNM, the dimensionality is still maintained, and the actual calculation time is probably slightly higher than 3.5×10^3 cycles. Next, the data flow through the FVG to generate the *query*, *key*, and *value* set with dimensions <4, 3, 9, 8>. The actual calculation time for each feature matrix is probably slightly higher than 2.1×10^4 cycles. After the query and critical flow through MAM-1 to obtain a feature similarity matrix with dimensions <4, 3, 9, 9>, this matrix and *value* flow through MAM-2 to obtain an attention matrix with dimensions <4, 9, 24>. The actual calculation time of the two steps is slightly higher than 8.1×10^3 cycles and 7.7×10^3 cycles, respectively. Finally, the attention matrix flows through the LM to obtain the output result with dimension <6, 6, 24>, and the actual calculation time is probably slightly higher than 2.1×10^4 cycles. By the thought about the throughput balance, we set the parallelism of these five stages to <1, 6, 3, 3, 6>, so that the time cost of each stage is around 3.5×10^3 cycles. Moreover, in theory, we can obtain an inference throughput performance of 2.86×10^5 inf/s at a running frequency of 100 MHz in FPGA for this example.

C. Norm Multilayer Perceptron Module

The structure of the NMP is shown in Fig. 5, which is composed of an LNM, a linear expander (LE), a non-LM (NLM), and a linear retractor (LR). The input data is the output calculated by NSA, standardized through the LNM. We can achieve partial output after passing through the LE, the NLM, and the LR. The calculation complexity of each part is shown in

$$\Omega(\mathrm{NMP}) = 2M_r * H * W * C^2 + M_r * H * W * C. \tag{5}$$

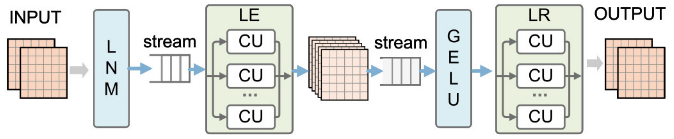

Fig. 7. Convert of input data through the calculation in NMP.

In NMP, each module also has different computing units to achieve different functions, and the detail of the calculation flow is shown in Fig. 7. The LNM is the same as NSA, and its overall calculation time is probably within the level of $H * W * C * 4$. Moreover, the LE expands the input M_r times in the feature dimension. Therefore, the overall calculation time of LE is probably within the level of $H * W * C^2 * M_r$. Then, the NLM uses the nonlinear function to convert input data. The ViT model frequently uses GELU as the nonlinear function. Furthermore, GELU needs to calculate the standard normal distribution transformation, which uses a large consumption of hardware resources. So, we separate this calculation section into a single computing unit to increase the efficient utilization of resources. This calculation time of NLM is probably within the range of $H * W * C * M_r$. Finally, the LR shrinks the input M_r times in the feature dimension. And its calculation time is in the same range as LE, $H * W * C^2 * M_r$.

Therefore, to ensure efficient processing in this stream computing structure, it is necessary to ensure that the throughput of each stage is similar. We make the throughput of the LNM as a baseline. The parallelism required by the LE should be $C * M_r/4$. The parallelism required by the NLM should be $M_r/4$. Finally, the parallelism of the LR should be $C*M_r/4$. In this way, it can ensure that we could achieve the overall computing performance with the full utilization of the computing power in this structure, as shown in

$$P_{\mathrm{NMP}} = \frac{\mathrm{Frequency} * \mathrm{FLOPs}}{\mathrm{Max}(\mathrm{Latency}_i)}, \quad i \in \mathrm{NMP}. \tag{6}$$

For example, assuming that the input dimension is <6, 6, 24>, the MLP ratio, M_r is 3, and the patch size is 3. After the input passes through the LNM, the dimensionality is still maintained, and the actual calculation time is probably slightly higher than 3.5×10^3 cycles. Next, the data flow through the LE to expand the input to output with dimensions <6, 6, 72>. The actual calculation time for each feature matrix is probably slightly higher than 6.2×10^4 cycles. Then, through the NLM, the data dimensionality is not changed, and the actual calculation time is probably slightly higher than 2.5×10^3 cycles. Finally, we obtain the output result with dimensions <6, 6, 24> after the input flows through the LR. Its actual calculation time is probably slightly higher than 6.2×10^4 cycles. By the thought about the throughput balance, we set the parallelism of these four stages to <1, 18, 1, 18>, so that the time cost of each stage is around 3.5×10^3 cycles. And in theory, we can obtain an inference throughput performance of 2.86×10^5 inf/s at a running frequency of 100 MHz in FPGA for this example.

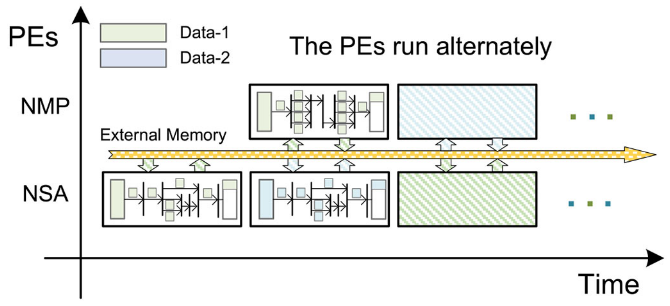

Fig. 8. Sequence diagram on the ViA. The NSA and NMP process data-1 and data-2 alternately when an application is running.

D. Data Flow

The whole calculation process is done collaboratively by NSAs and NMPs. Figs. 5–7 show that the NSA loads the input data from the external memory to the local buffer on the LNM. Then, the set of query, key, and value is then obtained after the buffered data passes through the LNM and FVG. Moreover, we cache the value set to the local storage on the MAM, while the computing unit in MAM calculates the first stage result of MSA. After the first stage, MAM loads the value set from the local storage. It processes the second stage of MSA and LM in sequence to obtain the intermediate result and write it to external memory. Meanwhile, when NSA is running, NMP also loads data from external memory to the local buffer on the LNM and processes the input data through LE, NLM, and LR to achieve the result of the whole transformer unit. By repeating this operation multiple times, the models with different layers and transformer units can be processed on the hardware accelerator.

The overall operation sequence for the ViA is shown in Fig. 8. In the beginning phase, data-1 is first loaded from external memory into the inside buffer of the NSA. After the internal streaming processing, data-1 completes the calculation in the upper half of the transformer. Data-1 is written back to external memory by the data write unit. And then, NSA and NMP simultaneously load data-2 and the data-1, compute in parallel with different works, and write back to external memory, respectively. The rest operation may be deduced by analogy. NSA and NMP will alternately process data-1 and data-2 until the end of the entire calculation process.

V. Method

This section introduces the overall design idea for ViA, the ViT accelerator. We explain the benefits of this architecture. The details are as follows.

A. Half-Layer Mapping

From the overall architecture design (Fig. 5), it can find that we split the entire transformer calculation process into two parts, NSA and NMP. The main reason is the path dependence problem caused by the shortcut mechanism. In the previous section, we explained that the shortcut mechanism allows us to train the deeper networks and solves the problem of network degradation. This fact thoroughly verifies the effectiveness of the shortcut mechanism, but the problem of path dependence in the calculation process cannot be avoided. This problem wholly affected the hardware pipeline design. Assume that for a P-level pipeline process, on common, $P-1$ buffer areas are needed to buffer intermediate data in the calculation. However, if it has a shortcut that spans the whole process, it needs to increase the $P-1$ buffer areas with the input data size to maintain the same performance before. This situation is often unacceptable for hardware platforms with limited on-chip resources.

However, according to the ViT, which also has the shortcut mechanism, the appearance of the shortcut calculation is very regular in the entire model. From the calculation flow of the ViT (Table II), it is easy to find that the shortcut calculation divides each transformer unit into two parts. This is the inside point of the half mapping method. As shown in Fig. 9, by mapping these two parts to two PEs, the residual unit is set at the beginning of the PE structure to calculate the residual of the previous layer. In this way, the path dependency problem caused by the original shortcut calculation can be avoided, and the overhead of on-chip resources can be reduced. As shown in (7), it expresses the improvement of the utilization efficiency of on-chip storage resources under the same computing

$$\text{Eff}_{\text{Buf}} = \frac{(B*H*W*C)*(P+1)*2}{(B*H*W*C)*(P+2)}. \tag{7}$$

Moreover, for an input data sample with a dimension of <6, 6, 24> and a patch size of 3, the storage size cost by the input data is $6*6*24 = 864$. Because of the path dependence caused by the residual structure, multistage buffers need to be used to reduce the computational overhead of pipeline waiting before half mapping optimization is performed. In the calculation process of the ViT, as shown in Fig. 8, when calculating the residual merging in the middle, the original input data need to be buffered through 5 stages. In other words, the additional storage overhead required is $4*864 = 3456$. Then when calculating the residual merging in the last, the intermediate results need to be buffered through four stages, and the additional storage overhead required is $3*864 = 2592$. After half mapping optimization, we can directly save the additional storage overhead, thereby eliminating the path dependence of the calculation process.

B. Data Partition

Parallel computing is a method to improve computing performance while fully using hardware resources, but the parallelism is always interrupted by some data correlations, reducing its operating efficiency. In the whole calculation process of the transformer, there is also the data correlation. As shown in Fig. 10, there are three parts in the ViT calculation, Layer Normalization, MSA, and MLP. From the workflow of these calculations, it can be seen that the original input data corresponding to each part of the output is slightly different. Therefore, we analyzed the independent correlation of input and output data in the whole calculation process, as shown in Table III. Moreover, independence means that the data can be divided on this dimension, and parallel processing is not affected; dependence means that after the data is divided on

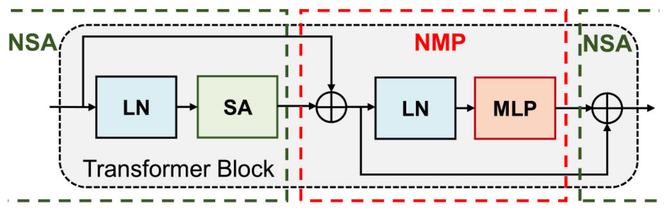

Fig. 9. Schematic of the half-layer mapping method.

TABLE III
DEPENDENCY ANALYSIS FOR DATA IN THREE CRITICAL PARTS OF THE VIT

Related Data		Input	Weight	Avg	Std	Output
Layer Norm	H	Ind	Irr	Ind	Ind	Ind
	W	Ind	Irr	Ind	Ind	Ind
	C	Ind	Ind	**D**	**D**	Ind
Multi-Heads Self-Attention	H	**D**	Irr	-	-	**D**
	W	**D**	Irr	-	-	**D**
	C	Ind	Ind	-	-	**D**
Multi-Layer Perceptron	H	Ind	Irr	-	-	Ind
	W	Ind	Irr	-	-	Ind
	C	Ind	Ind	-	-	Ind

this dimension, parallel processing will involve the influence of synchronization, merging, or computing order; irrelevant means that the calculation process of the data can be arbitrarily split and transformed on this dimension and will not affect its parallelism.

Therefore, if you want to perform computational splitting of the ViT, you need to avoid the dependency of the split dimensions. You must try to divide along the irrelevant and independent data dimensions. From the independent correlation of the input and output data of the whole calculation process shown in Table III, we divide it by the $<H, W, C>$ dimension. Convert the original data with dimensions of $<B, H, W, C>$ to $<W_S, W_S, C/N_{\text{head}}>$, so that the overall parallelism can be increased up to the $B*N*N$ times. Moreover, as shown in (8), there is a change of the data size of the cache on a single PE core in the theoretical

$$\begin{aligned} \text{Buffer} &= B*H*W*C*\text{precision} \\ &- > W_S^2 * C/N_{\text{head}} * \text{precision}. \end{aligned} \tag{8}$$

Moreover, for an input data sample with a dimension of $<6, 6, 24>$, the patch size is 3, and the multihead is 3, the storage size cost by the input data is $6*6*24 = 864$. Therefore, for the calculation of NSA and NMP, the storage size required for the entire calculation process is 7884 and 9504, respectively. After data partition optimization, we change the dimension of each input data to $<3, 3, 24>$, which is equivalent to reducing the storage size occupied by the input data to $3*3*24 = 216$. And then, this part of the input is processed in the kernel with the data flow. At this time, the storage sizes applied to the input on the NSA and NMP can be reduced to 1971 and 2376, respectively. After optimizing data partitioning, we can perform parallel processing on the partition dimension to achieve efficient acceleration calculations while making full use of hardware resources.

Algorithm 1: Optimization for Workload Homogeneity Based on the Greedy Strategy

```
Input: Data Size, Hardware Resources, Computing Unit
       Templates
Output: Implementation of the components
1  initialization;
2  CUs is the list of components' implement in NSA and
   NMP;
3  init(CUs.parallelism);
4  homo = CalHomogeneity(CUs);
5  old_homo = homo;
6  while homo <= old_homo do
7      $cu_i$ = findMaxLatency(CUs);
8      if canSpeedup($cu_i$) then
9          $cu_i$.parallelism+ =1;
10         CUs.update(genImplemente($cu_i$));
11     else
12         for $cu_k$ in CUs but not $cu_i$ do
13             decrease($cu_k$.parallelism);
14             CUs.update(genImplemente($cu_k$));
15         end
16     end
17     checkResource(CUs);
18     old_homo = homo;
19     homo = CalHomogeneity(CUs);
20 end
```

C. Design Space Exploration

Design space search (DSE) is inevitable for the design of the architecture. This section discusses our DSE approach for the proposed ViA architecture to obtain the optimal computational performance with the given hardware platform. From (2), (4), and (6), it can be found that the comprehensive performance of ViA is affected by the maximum latency of each computing part in the two PEs, NSA and NMP. Therefore, to optimize computing performance and hardware utilization, we use homogeneity to measure the results after optimization, as shown in

$$\begin{aligned} \text{Homogeneity} &= \frac{1}{n} * \sum_{i}^{n} \left(\text{Latency}_i - \text{Avg}(\text{Latency})\right)^2 \\ & i \in n * \text{CUs in NSA, NMP}. \end{aligned} \tag{9}$$

Therefore, to make the computing performance and hardware utilization better, we need to reduce the value of homogeneity as much as possible. Therefore, we propose a workload homogeneity algorithm based on the greedy strategy to complete the DSE task. As shown in Algorithm 1, after initializing parameters, we can obtain each computing unit component without parallelization from the hardware template. First, we find the component with the enormous latency in the current component list. Then, it determines whether the component can be accelerated based on the current situation to improve the computing performance and reduce the latency. After judgment, the component's internal configuration is modified, the parallelism increases by 1, and the computing unit component regenerates with the hardware template. For this component that can no longer be accelerated, the other part will be processed by reducing the parallelism of other components except this component and updating the entire component list. Furthermore, the algorithm checks the resource usage of the updated component list and updates the current and old values of the homogeneity. Finally, we will get the implementation of the computing unit with workload homogeneity according to the existing conditions.

VI. Evaluation

In this section, we implemented ViA on Xilinx Alveo U50 FPGA, which has a total of 5952 DSP slices, 1344 BRAMs, 872K LUTs, and 1743K FFs on this platform. We measure the overall performance on ViA by running the Swin Transformer in ImageNet and perform a comprehensive comparison with the result of CPU, GPU, and other FPGA accelerators.

A. Experiments Setup

In this work, our ViA is programmed on FPGA corresponding to the computation types using HLS-compatible C++, which forms the HLS core library with configurable parameters.

Baseline Models and Datasets: We test our method on a tiny version of Swin Transformer using the ImageNet-1K dataset. For this network model, there are four layers with various transformer blocks. Moreover, the Depth of layers is <2, 2, 6, 2>, the multihead H_A of transformer blocks is 3, and the factor M_r of MLP is 4.

Hardware and Toolkits: The hardware PEs, NSA and NMP, are synthesized and generated with Xilinx Vitis HLS 2021.2, and then we use Xilinx Vitis 2021.2 and Xilinx Vivado 2021.2 to synthesize and deploy the complete project. The target platform is Xilinx Alveo U50 FPGA Card to implement our ViA to perform the ViT model. Moreover, the host CPU controls the ViA accelerator and initial DDRs via the PCIe interface. The built-in power report in Vivado gives power consumption numbers.

B. PE Performance

In this section, we deployed the ViA architecture using the u50 board. Moreover, to verify the effectiveness of the optimization method, we perform the Tiny version of the Swin Transformer on ViA. After targeted optimization and deployment of the two processing engines, NSA and NMP, we test the performance and resource usage with/without our optimization, respectively.

1) Resource Utilization: In Table IV, we individually show the FPGA utilization results for several components of the NSA and NMP: the LNM, the FVG, the MAM, the LM, the LE, the NLM, and the LR. Moreover, we can get the cost of the pipeline by total utilization subtracting the number of all components. The number of BRAM in NSA and NMP is 104 and 88, respectively. Without half-layer mapping, this number increases ~2 times. Moreover, by the data partition, the size in each pipeline is reduced ~64 times.

TABLE IV
FPGA Resource Utilization for Components of NSA and NMP

PE	Module	BRAM	DSP	FF	LUT
NSA	LNM	2	7	2145	2817
	FVG	7	127	20824	20925
	MAM	25	131	32383	32783
	LM	7	42	7543	7756
	Total	145	307	67309	69018
NMP	LNM	2	7	2145	2817
	LE	7	168	26917	27107
	NLM	12	20	3161	3325
	LR	28	168	24763	22520
	Total	137	363	61352	59836

2) Compute Characteristic: For the Swin Transformer model, the input data dimensions of the first transformer layer are <56, 56, 96>, the number of long heads is 3, the linear widening size M_r is 4, and the patch size is 7. We design PEs, NSA, and NMP based on these parameters and optimize their computational throughput. As shown in Fig. 11, each processing part's original calculation amount and data amount are very different. Moreover, due to the existence of the shortcut mechanism, the pipeline operation between these components will consume 1304 BRAM or make the cost of calculation time up to 1 462 480 cycles, which is unacceptable for deployment on FPGA. After our half-layer mapping optimization strategy, it reduces the cost of BRAM to 145, which is about nine times and improves the overall computing performance by 64 times.

At the same time, we conducted a throughput analysis for the running instance. As shown in Fig. 10, the computational bottlenecks of NSA and NMP are the FVG and LE and LR parts, respectively, and the amount of computation is larger than other stages in the order of magnitude. For example, the FVG stage requires $1.4 * 10^6$ cycles, and the calculation homogeneity of each stage is $2.74344 * 10^{11}$ in NSA. This situation drags down the inference time of the model. After the throughput balance, we make the calculation time of each stage reduced to 22 387 cycles, which is 62.5 times higher than before. At the same time, the homogeneity of each stage is $7.2 * 10^6$, which speedup 38 089 times than before. Therefore, an output can be obtained every 22 387 cycles for a single data tile. Finally, the overall computing throughput of NSA and NMP can reach 157.8 fps.

C. Comparison With CPU and GPU

This section conducts a performance and energy efficiency comparison between CPU, GPU, and FPGA prototypes. We use Pytorch (v1.7.0) with CUDA 11.2 and cuDNN 7.6.5 to run model inference on both CPU and GPU, and the detailed information is listed as follows.

1) *CPU Baseline:* Intel Core i7-5930K processor with 15-MB cache, six physical cores, 12 threads operating at 3.7 GHz, and the thermal design power (TDP) is 140 W.

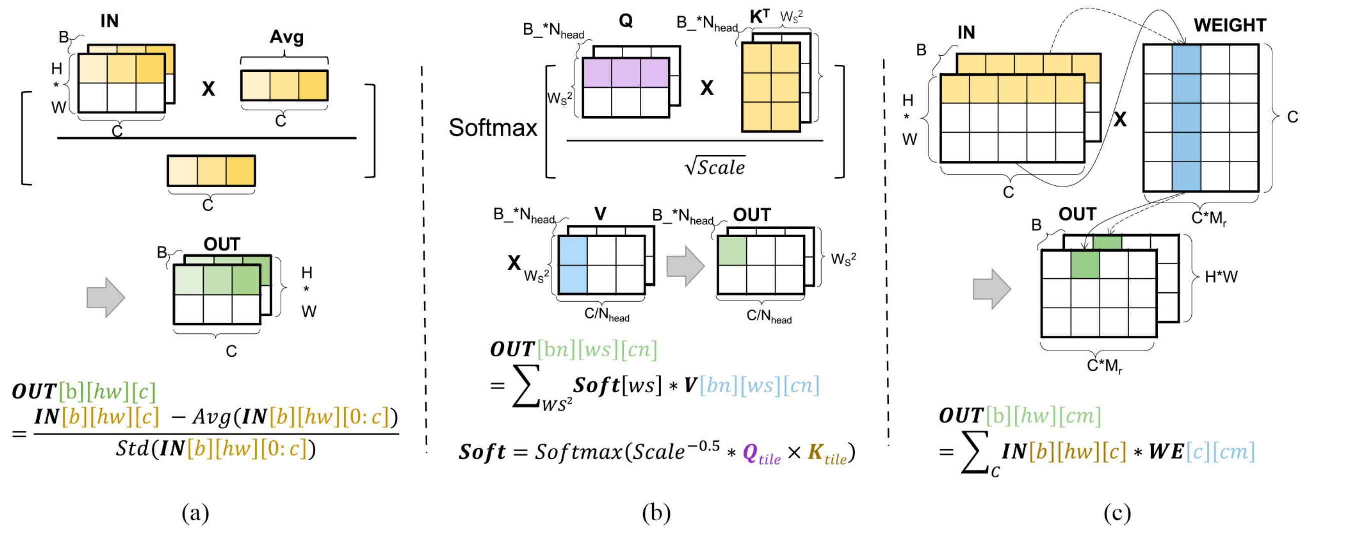

Fig. 10. Data analysis under the computing flow in three critical parts of the ViT. (a) Layer normalization. (b) Multiheads self-attention. (c) Full connection layer.

TABLE V
RELATIVE THROUGHPUT OF VIA COMPARED WITH CPU (I7-5930K), GPU (V100), AND FPGA (U50). WE ALSO GIVE RELATIVE THROUGHPUT COMPARED TO FT [18]

	CPU	GPU	FT [18]	ViA
Throughput	1X	72.0X	19.0X	59.5X
DSP Slices Utilized	-	-	4204	2420
Approximate Power	140	246	25	39
Performance/Power	0.007X	0.293X	0.76X	1.525X

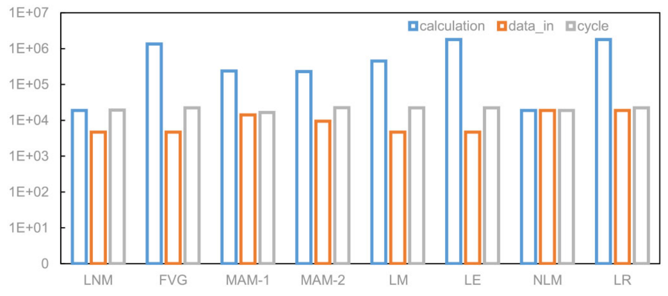

Fig. 11. Consumption of calculation and fan-in data in each component of NSA and NMP, and the latency of each component after optimization.

2) *GPU Baseline:* NVIDIA Tesla V100 with 32-GB GDDR5 and 5120 hardware threads operating at 1455 MHz.

We use the product of CPU utilization and TDP power to estimate the actual CPU power consumption, and the GPU runtime power is measured using nvidia-smi. The CPU and GPU versions run with the batch size set to 128. The Tesla V100 has 5120 compute units and runs at a 4.85× higher clock frequency than ViA. The inference performance and relative speedup are shown in Table V.

From Table V, we see that the comparison results, where all numbers are normalized to the CPU baseline. Compared to the CPU baseline, our prototypes achieve a 59.5× speedup on Swin Transformer in terms of performance. Compared to GPU, the performance improvement is 0.826×. Compared to the CPU, our prototypes achieve 217.86× speedup on Swin Transformer for energy efficiency. Compared to GPU, the energy efficiency improvement achieves 5.2× improvement. At the same time, we also give relative throughput compared to FT [18]. We use the theoretical performance in this article to compare with our ViA. Moreover, we achieve a 3.13× speedup in throughput and a 2.0× improvement in performance per power with less DSP than FT.

D. Comparison With Previous FPGA Accelerators

Because it is the past year that transformer be used in the CV, so up to now, there are few relevant publications for ViT accelerator. Therefore, in Table VI, we list some related work that design accelerator for transformer in NLP. To compare the TEC [17] with our work, data precision and model compression have a significant impact on performance and computational efficiency. The design of NPE [15] is the overlap. This pattern makes BRAM consumption less than the stream, but the cost is difficult to achieve high performance. The inference time in NPE is 232.37 ms when performing the BERT with sequence length 512. The throughput arrival is approximately the level of 100 GOPs. Therefore, our design achieves 3× higher throughput and 2.58× higher computational density. However, compared with the others, our design is higher 4× to 10× in throughput and higher 7× to 15× and 2.6× to 10.9× in calculation density and efficiency.

VII. RELATED WORK

The excellent performance of the transformer on NLP and CV attracts wide attention from researchers in the field of computer architecture. The design of structures for optimization based on the calculation characteristics of the transformer are also emerging endlessly. The first is to optimize the problem of excessive transformer calculations. TEC [17] has designed a work-balanced pruning method to make full use of the performance of the hardware platform. FT [18] uses

TABLE VI
PERFORMANCE COMPARISON WITH PREVIOUS FPGA IMPLEMENTATIONS

Related Work	Model/ Dataset	Specification			Power (W)	Resource utilization				Throughput (GOP/s)	Efficiency (GOP/j)
		Type	Frequency (MHz)	Precision		BRAM	DSP	LUT	FF		
TEC [17]	Multi30K	ZCU 102	150	INT8	18	912	2520	252K	161K	1870	103.8
FTRANS [19]	RoBERT	VCU 118	N/A	Fix16	25	N/A	6531	453K	509K	170	6.8
NPE [15]	BERT	Zynq Z-7100	200	16-bit	20	526	2019	156K	261K	N/A	N/A
FT [18]	TinyBert	Alveo U200	N/A	N/A	25	4204	4145	937K	504K	75.94	3.04
NMTA [20]	NMT	VCU 118	100	Float16	30	N/A	4838	556K	520K	22	0.73
CBBP [16]	Transformer	Alveo U200	N/A	N/A	25	N/A	2343	472K	378K	34.01	1.36
AT [49]	Transformer	Alveo U200	N/A	N/A	N/A	3244	3572	486K	604K	14.47	N/A
ViA[ours]	Swin T	Alveo U50	300	Float16	39	1002	2420	258K	257K	309.6	7.94

a mixed block and vector-wise method for pruning, which further reduces the computational complexity of the model. CBBP [16] even uses the column block pruning method to obtain a computing performance of 2091.8 fps. Moreover, FTRANS [19] uses the FFT/IFF method to reduce the matrix calculation complexity. Some researchers have taken a different approach and optimized the nonlinear function in the calculation process of the transformer to improve performance. ELSA [22] redesigned a hardware-friendly similarity measurement method to replace the softmax computing unit to improve overall performance. NPE [15] uses an irregular segment piecewise linear function estimation method to calculate nonlinear elements without loss of calculation accuracy.

VIII. CONCLUSION

In this article, we propose the ViA, a novel ViT accelerator architecture based on FPGA, to execute the transformer application efficiently. In data locality, we analyze the data structure in the ViT and design an appropriate partition strategy to reduce the impact of data locality in the image and improve the efficiency of computation and memory access. Meanwhile, In the aspect of path dependence, we observe the computing flow of the ViT and use the half-layer mapping and throughput analysis to reduce the impact of path dependence caused by the shortcut mechanism and fully utilize hardware resources to execute the transformer efficiently. In the stage of the experiment, we implement the ViA architecture in Xilinx Alveo U50 and finally achieved ~5.2 times improvement of energy efficiency compared with NVIDIA Tesla V100, and 4–10 times improvement of performance compared with related accelerators based on FPGA, that obtained nearly 309.6 GOP/s computing performance in the peek.

REFERENCES

[1] A. Vaswani *et al.*, "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 5998–6008.

[2] B. Wang, K. Liu, and J. Zhao, "Inner attention based recurrent neural networks for answer selection," in *Proc. 54th Annu. Meeting Assoc. Comput. Linguist.*, vol. 1, Aug. 2016, pp. 1288–1297. [Online]. Available: https://aclanthology.org/P16-1122

[3] S. Sukhbaatar, A. Szlam, J. Weston, and R. Fergus, "End-to-end memory networks," 2015, *arXiv:1503.08895*.

[4] T. Rocktäschel, E. Grefenstette, K. M. Hermann, T. Kočiskỳ, and P. Blunsom, "Reasoning about entailment with neural attention," 2015, *arXiv:1509.06664*.

[5] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," in *Proc. OpenAI blog*, vol. 1, 2019, p. 9.

[6] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proc. Conf. North Amer. Ch. Assoc. Comput. Linguist. Human Lang. Technol.*, vol. 1, Jun. 2019, pp. 4171–4186. [Online]. Available: https://aclanthology.org/N19-1423

[7] M. Masala, S. Ruseti, and M. Dascalu, "Robert—A romanian bert model," in *Proc. 28th Int. Conf. Comput. Linguist.*, 2020, pp. 6626–6637.

[8] A. Dosovitskiy *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," in *Proc. Int. Conf. Learn. Represent.*, 2021, pp. 1–21. [Online]. Available: https://openreview.net/forum?id=YicbFdNTTy

[9] I. Bello, B. Zoph, A. Vaswani, J. Shlens, and Q. V. Le, "Attention augmented convolutional networks," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2019, pp. 3286–3295.

[10] M. Cornia, M. Stefanini, L. Baraldi, and R. Cucchiara, "Meshed-memory transformer for image captioning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 10578–10587.

[11] P. Ramachandran, N. Parmar, A. Vaswani, I. Bello, A. Levskaya, and J. Shlens, "Stand-alone self-attention in vision models," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32. Red Hook, NY, USA: Curran Assoc., 2019. [Online]. Available: https://proceedings.neurips.cc/paper/2019/file/3416-a75f4cea9109507cacd8e2f2aefc-Paper.pdf

[12] H. Zhang, I. Goodfellow, D. Metaxas, and A. Odena, "Self-attention generative adversarial networks," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 7354–7363.

[13] Z. Liu *et al.*, "Swin transformer: Hierarchical vision transformer using shifted windows," 2021, *arXiv:2103.14030*.

[14] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.

[15] H. Khan, A. Khan, Z. Khan, L. B. Huang, K. Wang, and L. He, *NPE: An FPGA-Based Overlay Processor for Natural Language Processing*. New York, NY, USA: Assoc. Comput. Mach., 2021, p. 227. [Online]. Available: https://doi.org/10.1145/3431920.3439477

[16] H. Peng *et al.*, "Accelerating transformer-based deep learning models on FPGAs using column balanced block pruning," in *Proc. 22nd Int. Symp. Qual. Electron. Design (ISQED)*, 2021, pp. 142–148.

[17] X. Zhang, Y. Wu, P. Zhou, X. Tang, and J. Hu, "Algorithm-hardware co-design of attention mechanism on FPGA devices," *ACM Trans. Embedded Comput. Syst.*, vol. 20, no. 5S, pp. 1–24, 2021.

[18] P. Qi *et al.*, "Accelerating framework of transformer by hardware design and model compression co-optimization," in *Proc. IEEE/ACM Int. Conf. On Comput. Aided Design (ICCAD)*, 2021, pp. 1–9.

[19] B. Li *et al.*, "FTRANS: Energy-efficient acceleration of transformers using FPGA," in *Proc. ISLPED*, 2020, pp. 175–180. [Online]. Available: https://doi.org/10.1145/3370748.3406567

[20] Q. Li, X. Zhang, J. Xiong, W.-M. Hwu, and D. Chen, "Efficient methods for mapping neural machine translator on FPGAs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 7, pp. 1866–1877, Jul. 2021.

[21] S. Lu, M. Wang, S. Liang, J. Lin, and Z. Wang, "Hardware accelerator for multi-head attention and position-wise feed-forward in the transformer," in *Proc. IEEE 33rd Int. Syst. Chip Conf. (SOCC)*, 2020, pp. 84–89.

[22] T. J. Ham *et al.*, “ELSA: Hardware-software co-design for efficient, lightweight self-attention mechanism in neural networks,” in *Proc. ACM/IEEE 48th Annu. Int. Symp. Comput. Archit. (ISCA)*, 2021, pp. 692–705.

[23] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 2017. [Online]. Available: https://doi.org/10.1145/3065386

[24] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” 2014, *arXiv:1409.1556*.

[25] C. Szegedy *et al.*, “Going deeper with convolutions,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2015, pp. 1–9.

[26] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, “Densely connected convolutional networks,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 4700–4708.

[27] J. Hu, L. Shen, and G. Sun, “Squeeze-and-excitation networks,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 7132–7141.

[28] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, “Learning transferable architectures for scalable image recognition,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 8697–8710.

[29] M. Tan and Q. Le, “EfficientNet: Rethinking model scaling for convolutional neural networks,” in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 6105–6114.

[30] I. Radosavovic, R. P. Kosaraju, R. Girshick, K. He, and P. Dollár, “Designing network design spaces,” in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 10428–10436.

[31] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou, “Training data-efficient image transformers & distillation through attention,” in *Proc. Int. Conf. Mach. Learn.*, 2021, pp. 10347–10357.

[32] W. Wang *et al.*, “Pyramid vision transformer: A versatile backbone for dense prediction without convolutions,” 2021, *arXiv:2102.12122*.

[33] A. Azizimazreah and L. Chen, “Shortcut mining: Exploiting cross-layer shortcut reuse in DCNN accelerators,” in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, 2019, pp. 94–105.

[34] D. Balduzzi, M. Frean, L. Leary, J. Lewis, K. W.-D. Ma, and B. McWilliams, “The shattered gradients problem: If resnets are the answer, then what is the question?” in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 342–350.

[35] S. I. Venieris, A. Kouris, and C.-S. Bouganis, “Toolflows for mapping convolutional neural networks on FPGAs: A survey and future directions,” 2018, *arXiv:1803.05900*.

[36] W. Lou, L. Gong, C. Wang, Z. Du, and Z. Xuehai, “OctCNN: A high throughput FPGA accelerator for CNNs using octave convolution algorithm,” *IEEE Trans. Comput.*, vol. 71, no. 8, pp. 1847–1859, Aug. 2022.

[37] M. Dhouibi, A. K. Ben Salem, A. Saidi, and S. B. Saoud, “Accelerating deep neural networks implementation: A survey,” *IET Comput. Digit. Techn.*, vol. 15, no. 2, pp. 79–96, 2021.

[38] X. Wang, C. Wang, J. Cao, L. Gong, and X. Zhou, “WinoNN: Optimizing FPGA-based convolutional neural network accelerators using sparse Winograd algorithm,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 11, pp. 4290–4302, Nov. 2020.

[39] Y. Guan *et al.*, “FP-DNN: An automated framework for mapping deep neural networks onto FPGAs with RTL-HLS hybrid templates,” in *Proc. IEEE 25th Annu. Int. Symp. Field Programmable Custom Comput. Mach. (FCCM)*, 2017, pp. 152–159.

[40] C. Wang, L. Gong, Q. Yu, X. Li, Y. Xie, and X. Zhou, “DLAU: A scalable deep learning accelerator unit on FPGA,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 36, no. 3, pp. 513–517, Mar. 2017.

[41] V. Gokhale, A. Zaidy, A. X. M. Chang, and E. Culurciello, “Snowflake: An efficient hardware accelerator for convolutional neural networks,” in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, 2017, pp. 1–4.

[42] R. Hameed *et al.*, “Understanding sources of inefficiency in general-purpose chips,” in *Proc. 37th Annu. Int. Symp. Comput. Archit.*, 2010, pp. 37–47.

[43] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, “Optimizing fpga-based accelerator design for deep convolutional neural networks,” in *Proc. ACM/SIGDA Int. Symp. Field Programmable Gate Arrays*, 2015, pp. 161–170.

[44] Y. Gong *et al.*, “N3H-core: Neuron-designed neural network accelerator via FPGA-based heterogeneous computing cores,” 2021, *arXiv:2112.08193*.

[45] L. Gong, C. Wang, X. Li, H. Chen, and X. Zhou, “MALOC: A fully pipelined FPGA accelerator for convolutional neural networks with all layers mapped on chip,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 11, pp. 2601–2612, Nov. 2018.

[46] T. Geng, T. Wang, A. Sanaullah, C. Yang, R. Patel, and M. Herbordt, “A framework for acceleration of CNN training on deeply-pipelined FPGA clusters with work and weight load balancing,” in *Proc. 28th Int. Conf. Field Programmable Logic Appl. (FPL)*, 2018, p. 394.

[47] L. Gong, C. Wang, X. Li, and X. Zhou, “Improving HW/SW adaptability for accelerating CNNs on FPGAs through a dynamic/static co-reconfiguration approach,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 7, pp. 1854–1865, Jul. 2021.

[48] Z. Liu, Y. Dou, J. Jiang, and J. Xu, “Automatic code generation of convolutional neural networks in FPGA implementation,” in *Proc. Int. Conf. Field Programmable Technol. (FPT)*, 2016, pp. 61–68.

[49] P. Qi, Y. Song, H. Peng, S. Huang, Q. Zhuge, and E. H.-M. Sha, *Accommodating Transformer Onto FPGA: Coupling the Balanced Model Compression and FPGA-Implementation Optimization*. New York, NY, USA: Assoc. Comput. Mach., 2021, pp. 163–168. [Online]. Available: https://doi.org/10.1145/3453688.3461739