Design and Implementation of a Highly Parallel Systolic Array on FPGA

Hao Lu

Abstract—**With the rapid development of deep learning technology, deep neural networks have achieved remarkable achievements in fields such as computer vision, natural language processing, and speech recognition. The core lies in constructing multi-layer neural networks to learn and represent data, thereby accomplishing complex tasks such as image recognition, speech recognition, and natural language processing.This paper designs a highly parallel systolic array on FPGA to accelerate Vision Transformer networks. For the multi-head attention mechanism, a scalable and variable systolic array architecture is designed. This architecture significantly reduces data transmission latency by directly transferring data between processing units and achieves highly parallel computation. Besides，Designed a simple and efficient data flow for matrix multiplication operations.**

Index Terms—**FPGA, Matrix Multiplication, Systolic Array**

I. Introduction

The rapid advancement of artificial intelligence technology has accelerated the pursuit of custom AI chips[1]. These chips are specially optimized compared to traditional processors (such as CPUs), offering superior computational efficiency and energy use benefits for complex tasks, and demonstrating excellent performance in accelerating deep learning algorithms. AI chips not only facilitate the widespread application of AI technology in cloud computing and data centers but also play a crucial role in the field of edge devices, providing robust computational support in smartphones, wearable devices, and autonomous vehicles. Research and development of AI chips have a profound impact on technological progress and economic development, serving as a key driver for innovation in future intelligent systems.

Deep Neural Networks (DNNs) have become the cornerstone of modern AI applications, capable of distilling complex patterns and decision logic from vast amounts of data. Over time, the scale of DNNs and the volume of data they process have been growing, leading to an increased demand for computational resources during the training and inference processes of these large-scale networks. Consequently, the energy efficiency ratio[6] of DNNs has become a critical factor to consider in the design and deployment of AI systems. Against this backdrop, the development of AI chips that can effectively support DNN operations, especially accelerators for specific models like Convolutional Neural Networks (CNNs)[2] and Transformers[3], has emerged as an important direction to advance AI technology. Since its introduction in 2017, the Transformer model has made revolutionary progress in fields such as natural language processing (NLP). Its uniqueness lies in the adoption of the self-attention mechanism, which effectively captures long-distance dependencies within the input data, significantly enhancing model performance. As research has progressed, the Transformer model has been successfully applied to tasks such as image recognition and speech processing, demonstrating its potential beyond traditional deep learning models. The Vision Transformer (ViT)[4] model, in particular, has garnered widespread attention due to its outstanding performance across various visual tasks. However, the high demand for computational resources by the ViT model also presents new challenges, prompting researchers to explore more efficient model acceleration methods.

To address the computational challenges of the Transformer model, research and development of dedicated hardware accelerators have become a hot research area. Especially for ASIC (Application-Specific Integrated Circuit) accelerators, although they provide exceptional performance, their highly specialized design limits adaptability to new neural network architectures and is difficult to widely apply due to high development costs. Field-Programmable Gate Arrays (FPGAs) stand out for their unique programmability, offering flexibility that allows developers to reconfigure hardware resources to accommodate specific circuit functions as needed. For new algorithm designs, FPGAs offer an efficient path, enabling updates by simply adding new design elements to existing circuits without undergoing a complex hardware design process anew. Additionally, FPGAs inherently possess parallel processing capabilities and low-power characteristics, making them an ideal choice for applications requiring rapid iteration and strict energy efficiency. CNNs and Transformer models hold a central position in the field of deep learning and artificial intelligence. Moreover, some deep learning networks employ both convolutional layers and self-attention mechanisms, so developing an FPGA accelerator capable of effectively accelerating both types of models would have significant value in improving energy efficiency and reducing computational costs.

In recent years, researchers have begun exploring unified hardware accelerators capable of simultaneously accelerating Transformer and CNN models. This type of accelerator design aims to be compatible with these two mainstream deep learning architectures. For convolution and attention mechanisms, Li[5] and others implemented a unified FPGA-based accelerator, with the proposed unified

Manuscript received March 17, 2024

Hao Lu, School of Software Engineering, Tiangong University, Tianjin, China

accelerator's computational efficiency surpassing the most advanced Transformer and ResNet-50 accelerators by 11.86% and 28.29%, respectively. Considering the above, designing a high-performance Transformer accelerator is crucial. The main contributions of this work are as follows:

1) An efficient data flow method for matrix multiplication has been designed, thereby accelerating the inference speed of neural network.

2) A scalable and variable systolic array architecture has been designed and implemented, enabling high parallelism in computations.

3) VIT-tiny algorithm achieves excellent performance when implemented on FPGA.

II. Related Work

A. *Vision Transformer*

Before ViT, CNNs were almost the dominant architecture for image recognition and visual tasks. Part of the success of CNNs is attributed to their ability to efficiently capture local structural information of images through local receptive fields and weight sharing mechanisms, which somewhat resembles the way the human visual system processes information. However, CNNs also face some limitations in processing images, such as difficulty in capturing long-distance dependencies, and a sharp increase in model complexity and computational cost with increasing network depth. ViT, through its self-attention mechanism, can establish direct associations between any two points in a sequence, allowing the model to have a global receptive field from the beginning and dynamically adjust its focus according to the task, concentrating on relevant areas of the image. Furthermore, ViT performs exceptionally well on large-scale datasets, with its performance improving as the amount of data increases, which is a clear advantage in data-rich application scenarios.

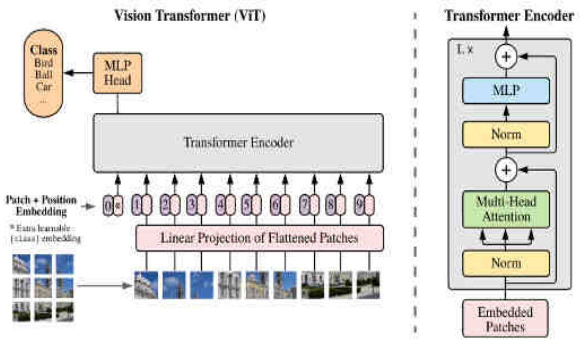

Figure 1. Network Diagram of ViT

Figure 1 depicts the network structure of the Vision Transformer (ViT). Initially, input feature maps are processed through a convolutional layer (Conv2d), which acts as the encoding process from images to patches, dividing the image into a series of small segments. Next, through a flattening operation (Flatten), the two-dimensional patches are transformed into a one-dimensional sequence. These sequences are then concatenated (Concat) with a class token. Subsequently, position embedding is added to the sequence to provide positional information for each element within the sequence. At this stage, it is evident that there is a need for acceleration in hardware for convolution operations, and Concat and Add operators, as well as a need for designing a matrix tiling form of data flow.

Afterward, the sequence data enters one or multiple Transformer encoder blocks (Encoder Block), each comprising four parts. The first part is layer normalization (LayerNorm), which normalizes the input to accelerate training and enhance the model's stability. The second part is the multi-head self-attention mechanism (Multi-Head Attention), allowing the model to focus on different parts of the input at various positions simultaneously. The third part is the residual connection. The fourth part consists of a multilayer perceptron (MLP), which includes linear layers and activation functions (such as GELU).

B. *Systolic Array*

A Systolic Array is an efficient parallel computing architecture originally proposed by H.T. Kung and Charles Leiserson[6] in the early 1980s. It is designed to accelerate dense matrix and vector operations, particularly in the fields of digital signal processing and deep learning. The term "systolic array" comes from the way data flows between processing elements (PEs) in a manner similar to the pulsation of blood in the heart. Data and operation instructions flow between the processing elements in the array at a fixed rhythm, enabling efficient data movement and computation.

In the hardware implementation field of matrix multiplication, the systolic array, as a classic design, offers two mainstream implementation modes: weight-fixed [7] and output-fixed [8]. In the weight-fixed systolic array, as illustrated in Figure 2, weight parameters are preloaded into each processing element (PE) before computation begins. This approach enables stable reuse of weights during computation, reducing the need for data movement. Additionally, continuous calculations can be performed as data flows through the processing elements, thus improving efficiency.

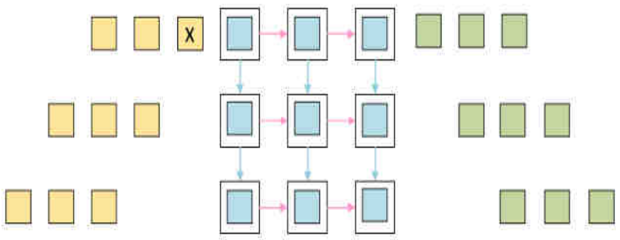

Figure 2. The weight-fixed systolic array

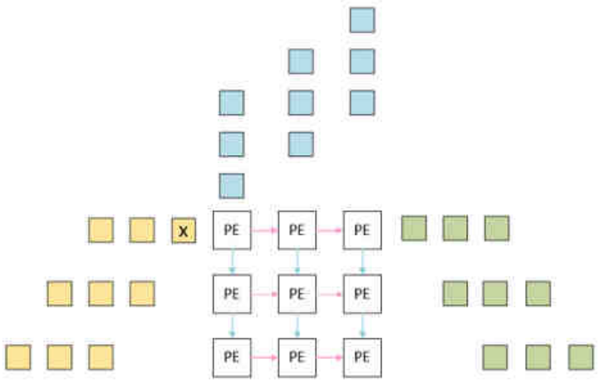

Figure 3 The output-fixed systolic array

Another mode is the output-fixed systolic array, depicted in Figure 3, where the transmission mode of input x remains unchanged, i.e., it is passed to the next PE once per clock cycle. However, unlike the weight-fixed mode, weight parameters are not statically loaded but dynamically flow, similar to input x, into the PEs for computation. This mode allows for more flexibility, particularly useful in scenarios with frequent weight updates, as it avoids the time and energy overhead required for reloading the entire weight matrix.

Many accelerator core computational modules currently use systolic arrays for computation. In 2021, He et al. [9] designed and implemented a CNN accelerator based on RISC-V, with the core module utilizing a systolic array. Additionally, Xu et al. [10] proposed a heterogeneous systolic array architecture for compact CNN hardware accelerators. In 2023, Chen et al. [11] developed a high-frequency systolic array in an FPGA-based Transformer accelerator. For MHA and FFN, they achieved working frequencies of 588 MHz and 474 MHz, respectively, on different-sized arrays. The generator designed in this paper mainly generates two-dimensional systolic arrays for accelerator use.

III. Implementation

A. *overall architecture*

Figure 4 illustrates a framework for implementing a ViT accelerator on an FPGA. The host PC or the processing system (PS) of Zynq serves as the host, controlling the FPGA through the PCIe interface. It is responsible for sending data to DDR memory and issuing instructions to the instruction registers inside the FPGA. This design allows the host to finely control the entire accelerator, including the data processing flow and parameter settings. The DMA module efficiently reads data from DDR based on instructions from the host and transfers it to the computation module. Simultaneously, the DMA is responsible for writing back processed data from the computation module to DDR memory. This process greatly enhances data processing efficiency by reducing CPU intervention and facilitating direct data transfer between memory and computation units.

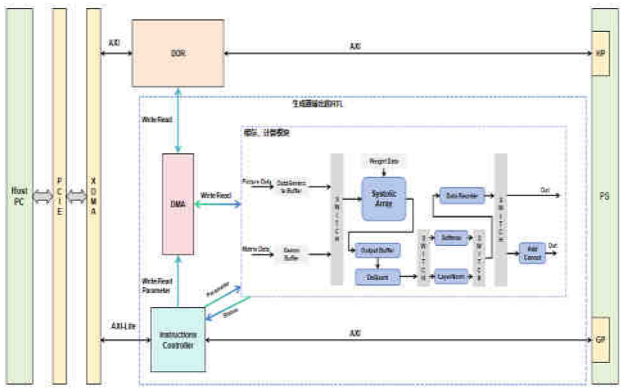

Figure 4 overall architecture of the ViT accelerator

Firstly, the feature image data is transmitted from the Host to the FPGA and cached into the DataGenerate Buffer module. The DataGenerate Buffer serves as a temporary storage area to receive and organize image data, converting it into matrix format suitable for subsequent processing. This step ensures that the data can be efficiently handled by the systolic array. If the Embedding layer is not executed, there is no need to enter this module; the data can simply flow into the matrix cache module.

Next, the processed matrix data stream, along with the corresponding weight data, is fed into the systolic array, where rapid data processing is achieved through its highly parallel structure. Here, each input data element undergoes multiplication with the corresponding weight, and the results are accumulated to produce the final output. Due to the parallel nature of the systolic array, this step significantly enhances computational throughput and efficiency. The computed data is then sent to the Output Buffer, another buffer for storing intermediate results generated by the systolic array. This buffer not only smooths the data flow to subsequent modules but also ensures that the processing speed matches that of the backend modules, preventing potential data congestion.

Secondly, the computed data enters the de-quantization module, where the output results are re-transformed to 8-bit after the quantization step. After completing the quantization step, the data reaches the Switch Selection module. This module acts as the commander of data routing, determining which processing unit the next step data should flow to based on the instruction registers. Here, the data is meticulously routed, either entering the Softmax unit along the path to be transformed into a distribution representing category probabilities or flowing to the LayerNorm unit for normalization to ensure consistency and stability of the network layer outputs.

Finally, the data enters the data reload module, responsible for transforming the rows and columns of the matrix to meet different data arrangement requirements. Finally, depending on the need, the data may undergo additional processing through Add, Concat modules.

B. *Systolic Array of architecture*

Figure 5 depicts an architecture design of a systolic array, widely employed in parallel computing and particularly in hardware accelerators for deep learning. The left side of the diagram illustrates the entire systolic array, while the right side shows the internal structure of a single processing element (PE). The array on the left consists of multiple PEs represented by boxes arranged in a matrix formation. Each PE is interconnected with its surrounding PEs, allowing data to flow between them. At the top of the systolic array, there is a group labeled "weight" serving as input, where each PE can receive weight data or matrix B data. Corresponding to the data inputs, there is a "w_valid" signal indicating the validity of the respective weight or matrix data. On the left side of the systolic array, there are inputs labeled "activate" and a "a_valid" signal, representing the input of the activation matrix and its validity, respectively. On the right side of the array, there is an output labeled "out," representing the output after data processing. Each PE has an output, and the results of each row flow out on the right side, similar to the connections of the last row of PEs in the array, as depicted in the diagram.

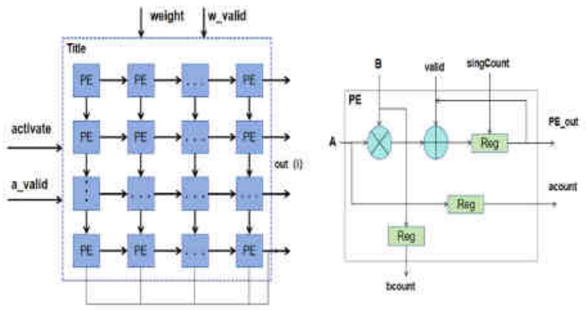

Figure 5 the architecture of Systolic Array

The structure on the right side illustrates the internal composition of a single PE. The PE contains a multiplier and an adder. Inputs to the multiplier are labeled "A" and "B," indicating that it performs multiplication operations on A and B. The output of the multiplier is connected to an adder, which is then connected to a register (Reg) used to temporarily store intermediate results. Another input to the adder is the output of another register, indicating that some data may need to be held before addition operations. Additionally, there is a "valid" signal and a "singCount" signal input to this PE, which may be used for synchronization and counting functions. Finally, the output of the PE is labeled "PE_out," "account," and "bcount," where "account" and "bcount" may represent the outputs of two different types of counters or registers.

C. Matrix Multiplication Operation

The computational mode employed in this paper is output-fixed for the systolic array, allowing it to perform matrix multiplication of arbitrary dimensions. In this mode, the data formats of matrices A, B, and C are all stored row-wise. This format offers the advantages of simplicity and ease of accessing input and output data formats. As a result, the design and implementation of the input control modules for matrices A and B become simpler, significantly reducing the workload of designing the data arrangement modules. As shown in Figure 6.

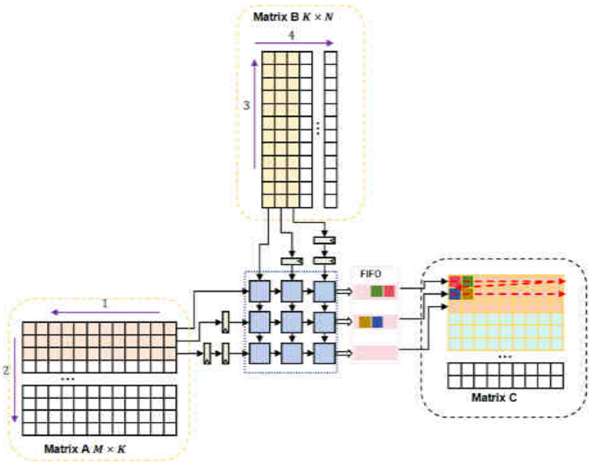

Figure 6 Matrix multiplication computation diagram

As shown in Figure 7, the left matrix cache employs a dual-BRAM strategy, allowing data reading and transmission to proceed in parallel through ping-pong operations. While the computation module processes data in the first buffer, the system can simultaneously write new data to or read from the second buffer. For the computation of matrices $A_{m\times k}$ and $B_{k\times n}$, assuming an 8×8 array, the first 8 rows of matrix A are initially loaded into Bram1. If the width of the systolic array is greater than n of matrix B, the calculation can be completed in one go. If it's smaller than n, the data in Bram1 needs to be cyclically read out, with the number of reads being n/8. Meanwhile, data is loaded into Bram2 while Bram1 is being read. After Bram1 data is read, data is then read from Bram2. The dual buffering ensures a continuous supply of data to the computation module, avoiding intermittent data flow due to buffer waiting for availability. This dual-BRAM approach reduces waiting time and improves processing speed. The RW state machine controls when to write data from DDR to BRAM and when to read data from BRAM to the array. The BramJudge state machine controls the read and write states of the two BRAMs to prevent data flow interruption. For the cache of the matrix above the array, n BRAMs are used for caching, where n is determined by the width of the array, and the address controller fetches data from different BRAMs.

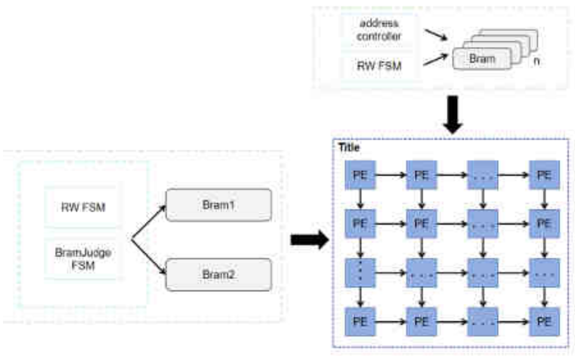

Figure 7 Matrix multiplication cache

IV. Experiment And Result

This experiment compares the inference acceleration of ViT-Base, to validate the flexibility and versatility of the accelerator designed by the generator. CIFAR-10 dataset is chosen for the experiment, which is a widely used benchmark dataset for image recognition and classification. The dataset consists of 60,000 32x32-pixel color images categorized into 10 classes, with each class containing 6,000 images. These classes include airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships, and trucks. Due to the lower pixel resolution of the images, they are first resized to 224×224.

The experiment utilizes the AX7Z100 board, employing an overall architecture of ARM+FPGA. Image weights are transferred from the PS side to the PL side through the HP (High-Performance) port, while instructions are transferred to the register module through the GP (General-Purpose) port. The PS side utilizes Xilinx's SDK, while the PL side is built using Vivado.In this experiment, Zynq board, adopting an overall architecture of ARM+FPGA, is employed. Image weights are transferred from the HP port on the PS side to the PL side, while instructions are transmitted to the register module via the GP port. Xilinx's SDK is used for the PS side, and Vivado is used for the PL side.

Table 1 Comparison of performance between different arrays

	Size of Array	
	8×8	8×24
LUT	25200	35407
LUTRAM	4087	3894
FF	34454	42265
BRAM	564	562.5
DSP	201	329
Performance (GOP/s)	25	75
Power(W)	4.5	5.4

In Chapter 3, the architecture of the systolic array was introduced. The size of the array is adjustable, and experiments were conducted with array sizes of 8x8 and 8x24 for accelerating ViT inference. Power consumption and performance were tested. As shown in Table 1,the accelerator achieved low power consumption and good performance.

V. Conclusion

Existing FPGA-based deep learning accelerators primarily focus on optimizing on-chip computing modules. These solutions concentrate on optimizing opportunities for parallel computation of feature input/output and convolution kernels. They utilize a large number of FPGA computing units based on various computational patterns. Although significant speed improvements have been achieved, surpassing CPU performance, these designs still lead to power consumption and resource wastage. Other design approaches emphasize internal and external communication to achieve system-level high throughput and maximize data reuse, thereby optimizing bandwidth. However, they do not fully exploit the computational capabilities of FPGAs. Therefore, the right accelerator architecture becomes crucial, not only to provide excellent computational power but also to consider the degree of data reuse, minimizing the pressure on IO operations to the greatest extent.

Based on the aforementioned issues, this paper designs a highly parallel systolic array architecture. With the systolic array structure as the computational core, the architecture's regular layout and localized interaction enhance the accelerator's computational throughput, while reducing data IO operations, thereby alleviating the bandwidth pressure on the AXI bus and reducing resource consumption.

References

[1] Momose, Hiroshi, Tatsuya Kaneko, and Tetsuya Asai. "Systems and circuits for AI chips and their trends." Japanese Journal of Applied Physics 59.5 (2020): 050502.

[2] Khan A, Sohail A, Zahoora U, et al. A survey of the recent architectures of deep convolutional neural networks[J]. Artificial intelligence review, 2020, 53(8): 5455-5516.

[3] Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need[J]. Advances in neural information processing systems, 2017, 30.

[4] Dosovitskiy A, Beyer L, Kolesnikov A, et al. An image is worth 16x16 words: Transformers for image recognition at scale[J]. arXiv preprint arXiv:2010.11929, 2020.

[5] Li, Tianyang, et al. "Unified accelerator for attention and convolution in inference based on FPGA." 2023 IEEE International Symposium on Circuits and Systems (ISCAS). IEEE, 2023.

[6] Kung, Hsiang-Tsung. "Why systolic architectures?." Computer 15.1 (1982): 37-46.

[7] Farabet, Clément, et al. "Neuflow: A runtime reconfigurable dataflow processor for vision." CVPR 2011 workshops. IEEE, 2011.

[8] Du, Zidong, et al. "ShiDianNao: Shifting vision processing closer to the sensor." Proceedings of the 42nd annual international symposium on computer architecture. 2015.

[9] He, Yangyang. "Design and implementation of convolutional neural network accelerator based on RISCV." Journal of Physics: Conference Series. Vol. 1871. No. 1. IOP Publishing, 2021.

[10] Xu, Rui, et al. "Heterogeneous systolic array architecture for compact cnns hardware accelerators." IEEE Transactions on Parallel and Distributed Systems 33.11 (2021): 2860-2871.

[11] Chen, Yonghao, et al. "High-frequency systolic array-based transformer accelerator on field programmable gate arrays." Electronics 12.4 (2023): 822.