CSTrans-OPU: An FPGA-based Overlay Processor with Full Compilation for Transformer Networks via Sparsity Exploration

Yueyin Bai*, Keqing Zhao, Yang Liu, Hongji Wang, Hao Zhou, Xiaoxing Wu, Jun Yu and Kun Wang†
School of Microelectronics, Fudan University, Shanghai, China
Email: *20112020048@fudan.edu.cn, †kun.wang@ieee.org

ABSTRACT

A few overlay processors for transformer networks emerge to achieve reconfigurable architectures and dynamic instructions. However, these processors consistently neglect exploring network sparsity, while existing sparse accelerators inefficiently utilize resources with separate computation parts. Furthermore, mainstream compilers for instruction generation are intricate and demand significant engineering efforts. In this work, we propose *CSTrans-OPU*, an FPGA-based overlay processor with full compilation for transformer networks via sparsity exploration. Specifically, we customize a multi-precision processing element (PE) array with DSP-packing for unified computation format with full resource utilization. Additionally, the introduced sorting and computation mode selection modules make it possible to explore the token sparsity. Moreover, equipped with a user-friendly compiler, *CSTrans-OPU* enables model parsing, operation fusion, model quantization, instruction generation and reordering directly from model files. Experimental results show that *CSTrans-OPU* achieves 6.92-20.06× speedup and 182.48× higher energy efficiency compared with CPU, and 1.47-3.85× latency reduction with 4.63-52.53× better energy efficiency compared with GPU. Furthermore, we observe up to 4.28× better latency and 4.94× higher energy efficiency compared with previously customized accelerators, and can be up to 1.93× faster and 4.39× more energy efficient than FPGA processors. To the best of our knowledge, our *CSTrans-OPU* is the first overlay processor for transformer networks considering sparsity.

ACM Reference Format:
Yueyin Bai*, Keqing Zhao, Yang Liu, Hongji Wang, Hao Zhou, Xiaoxing Wu, Jun Yu and Kun Wang†, School of Microelectronics, Fudan University, Shanghai, China, and Email: *20112020048@fudan.edu.cn, †kun.wang@ieee.org. 2024. CSTrans-OPU: An FPGA-based Overlay Processor with Full Compilation for Transformer Networks via Sparsity Exploration. In *61st ACM/IEEE Design Automation Conference (DAC '24), June 23–27, 2024, San Francisco, CA, USA*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3649329.3657325

†Corresponding author
This work was financially supported in part by National Key Research and Development Program of China under Grant 2021YFA1003602, in part by Shanghai Pujiang Program under Grant 22PJD003.

DAC '24, June 23–27, 2024, San Francisco, CA, USA

ACM ISBN 979-8-4007-0601-1/24/06…$15.00
https://doi.org/10.1145/3649329.3657325

1 INTRODUCTION

Transformer network has attracted a lot of interest from both academic and industry researchers. By leveraging self-attention mechanism, it can potentially replace convolutional neural network (CNN) and recurrent neural network (RNN) in multiple scenarios. Deploying transformer network on field-programmable gate array (FPGA) has also become a research trend recently, as FPGA strikes an effective balance among massive parallelism, high energy efficiency and short development cycle.

According to above advantages, there are multiple accelerators [9, 10] to implement attention mechanism or transformer networks on FPGA. These works always customize specific circuits for individual networks, while optimizing pipelines or reducing computation load by approximation or compression methods. Nevertheless, most of them suffer from fixed data flows. In other words, when the network topology is changed or updated, they cannot support the modified structures and need to be re-designed. To increase the flexibility of the architecture, a few FPGA-based processors [6–8] emerge with overlay architectures to support multiple transformer networks. However, these works are compatible to limited structures, NPE [8] only evaluates on different scales of BERT [4] networks while Vis-TOP [6] is restricted to various Swin [2] networks. HPTA [7] also performs evaluations only on BERT [4] and ViT [5] networks.

It is crucial to emphasize that current processors are tailored for transformer networks operating under dense structures. However, the consideration of model sparsity is essential due to redundant computation load in transformer networks. In this context, token sparsity is already explored in several accelerators [13, 16]. Nevertheless, aforementioned works lack the ability to support calculations of different sparsity by a unified structure. For instance, when the transformer network operates in a high-precision mode and requires dense operations merely, the hardware resources dedicated to sparse attention remain to be idle [13]. The inefficiency in resource utilization is further amplified as one DSP block is always leveraged for single multiplication.

Moreover, an end-to-end compiler is indispensable for constructing an overlay processor. Specifically, the compiler generates dynamic instruction streams so that underlying data paths can be easily changed according to detailed requirements. However, mainstream compilers are crafted for various neural networks, rendering them to be complex and user-unfriendly, demanding significant efforts for the integration of custom back-ends.

To address above issues, we propose *CSTrans-OPU*, an FPGA-based overlay processor with full compilation for transformer networks via sparsity exploration. Specifically, a multi-precision processing element (PE) array with DSP-packing unifies the computation with sufficient resource utilization. The sorting and mode

selection modules also enable us to explore the sparsity while a user-friendly compiler for transformers is further introduced.

The contributions of our work are as follows:

• **A Flexible Overlay Architecture.** In order to facilitate the acceleration of diverse transformer networks under varying sparsity degrees, we introduce a reconfigurable overlay architecture. This architecture is flexible enough according to dynamic data flows controlled by our instruction set architecture.

• **Sparsity Exploration with Sufficient Resource Utilization.** To perform computation of different sparsity degrees by a unified structure, we introduce a series of multi-precision PEs in *CSTrans-OPU*. The PEs are based on DSP-packing technology for sufficient hardware resource utilization. The sorting and mode selection modules are also included in *CSTrans-OPU* for sparsity exploration to different transformer networks.

• **Full Compilation Support.** We dedicate a compiler for transformer networks in *CSTrans-OPU*. The compiler is simple yet user-friendly, containing functions of model parsing, operation fusion, model quantization, instruction generation and reordering.

• **Competitive Performance.** Experimental results show that our design is rather competitive. *CSTrans-OPU* achieves 6.92-20.06× speedup and 182.48× higher energy efficiency compared with CPU, and 1.47-3.85× latency reduction with 4.63-52.53× better energy efficiency compared with GPU. Furthermore, we observe up to 4.28× better latency and 4.94× larger energy efficiency compared with previous accelerators and can be up to 1.93× faster than FPGA processors with 4.39× higher energy efficiency.

2 BACKGROUND

Transformer Networks. In recent years, transformer networks have drawn a surge of interests from the deep learning community. By stacking multiple encoder and decoder layers with attention modules, they have been proposed and demonstrated superior performance over traditional deep neural networks. Several outperforming transformer networks, *i.e.*, BERT [4], GPT [14], ViT [5] and Swin [2] have been applied to various domains. Despite the significant performance improvement, transformer networks are always hard to be deployed on resource-constrained devices due to the large model size and high computation cost [3].

Token Sparsity. Many previous works have proved that there are redundancy in the input sequence of transformer networks [13, 16]. Specifically, in attention computation, tokens only attend their nearby tokens and a few sampled tokens as the summary of the sentence instead of attending all tokens. As the time consumption of self-attention increases with the number of input tokens, the less important tokens are likely to be pruned to reduce computation load. In this way, the pruned tokens are always removed directly or replaced by a low-precision representation, while the other tokens remain in a high-precision format.

3 HARDWARE ARCHITECTURE

3.1 Architecture Overview

The overview of our *CSTrans-OPU* is shown in Fig.1. The architecture is composed of two sides, *i.e.*, software side and hardware side. On the software side, there is a compiler responsible for model parsing, network optimization and instruction generation, which is detailed in Section 4. On the hardware side, we propose a reconfigurable hardware architecture. Using multi-precision PEs with DSP-packing and sorting and mode selection modules, we can explore token sparsity of transformer networks with unified computation format and full resource utilization.

3.2 Multi-precision PE with DSP-packing

In our matrix multiplication unit (MMU), we use a series of multi-precision PEs with DSP-packing technology. Since tokens only attend their nearby tokens and a few sampled tokens as the summary of the sentence, most of the probabilities after softmax function are close to 0. Therefore, unsigned 4-bit integers rather than signed 8-bit integers can be used to represent their values. Therefore, our PE is designed as multi-precision to satisfy the 4-bit multiplication after softmax function and 8-bit multiplication in other computation parts. To further enhance the optimal utilization of hardware resources, we incorporate the DSP-packing technology in our multi-precision PEs to enable simultaneous execution of multiple signed multiplications in a single DSP block.

8-bit PE with DSP-packing. For two 8-bit signed-signed multiplication with a common factor, *e.g.*, $A \times C$ and $B \times C$, the architecture of the PE with DSP-packing technology is shown in Fig.2(a). The operands A and B are packed in the 27-bit input port of the DSP block, and the common factor C is packed in the 18-bit input port. However, the basic DSP-packing technology does not support signed-signed multiplication. To solve this problem, for the computation of $A \times C$, when C is a signed integer, the correct product is obtained by adding "−1" to the basic result. For $B \times C$, since the sign bit of B is not the most significant bit (MSB) of the packed A and B, B is treated as an unsigned 8-bit integer B^*, resulting in the incorrect product at the lower 16 bits. The computation of $B \times C$ is shown in Eq.(1), where B is represented by $b_7 b_6 ... b_0$.

$$\begin{aligned} B \times C &= (-b_7 \times 2^7 + \sum_{i=0}^{6} b_i 2^i) \times C \\ &= (-b_7 \times 2^7 \times 2 + B^*) \times C \\ &= -b_7 \times 2^8 \times C + B^* \times C \end{aligned} \tag{1}$$

For $B \times C$, the product can be corrected using only one addition, which can be realized by the accumulation component in DSP block. In our PE, the added part is called error compensation(EC). As illustrated in Eq.(1), the $-b_7 \times 2^8 \times C$ part is the EC for $B \times C$. While for $A \times C$, EC is "1" when C is a signed integer.

4-bit PE with DSP-packing. For the 4-bit multiplication after softmax function, the architecture of the PE with DSP-packing is shown in Fig.2(b). The four 4-bit operands represented as A, B, C and D, are packed in the 27-bit input port of DSP block, and a common 4-bit operand represented as E, is packed in the 18-bit input port. In order to make each of the products separated, the four operands A, B, C, and D must be at least 4 bits apart. For the three operands B, C and D, they are also treated as unsigned 4-bit integers in DSP block, so we need to add their corresponding ECs to get the correct products. Since adding ECs to $B^* \times E$, $C^* \times E$ and $D^* \times E$ does not generate a carry in our 4-bit PE, adjacent products do not overlap. The difficulty of 4-bit multiplication is that the MSB of A cannot be packed into the input port of DSP block. Consequently, A is treated as a signed 3-bit integer, $\widetilde{A}$, resulting in an incorrect

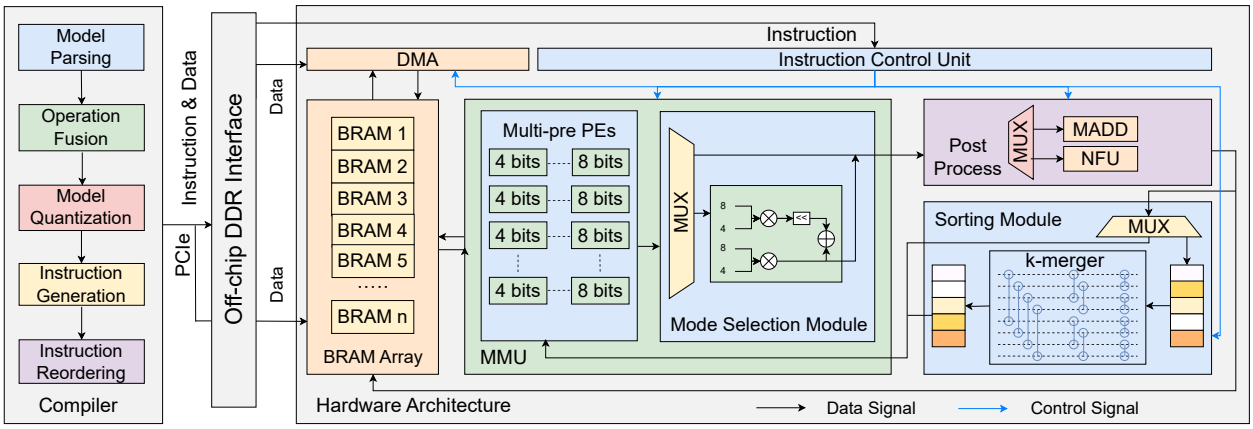

Figure 1: The overview of *CSTrans-OPU* overlay architecture.

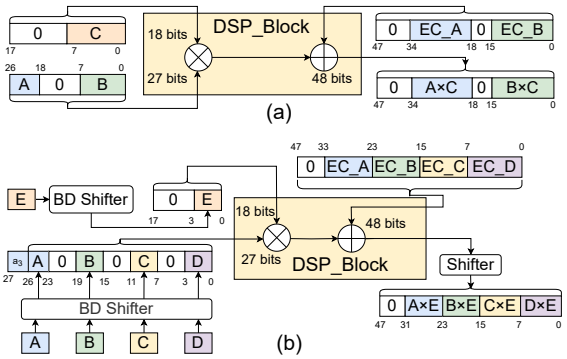

Figure 2: The structure of our multi-precision PEs with DSP-packing technology. (a) 8-bit PE; (b) 4-bit PE.

product. To address this difficulty, we use the shifter, bit-wise AND, an addition operation and a subtraction operation to correct the product. The computation of $A \times E$ is shown in Eq.(2), where A is represented by $a_3a_2a_1a_0$ and $\widetilde{A}$ is represented by $a_2a_1a_0$. Note that the $(-a_3 + a_2) \times 2^3 \times E$ in Eq.(2) is the EC for $A \times E$.

$$\begin{aligned} A \times E &= (-a_3 \times 2^3 + \sum_{i=0}^{2} a_i 2^i) \times E \\ &= (-a_3 \times 2^3 + a_2 \times 2^3 + \widetilde{A}) \times E \\ &= (-a_3 + a_2) \times 2^3 \times E + \widetilde{A} \times E \end{aligned} \tag{2}$$

Note that some probability after softmax function is not close to 0. Instead of directly truncating 4-bit from the MSB, we design a shift unit with leading bit detection (*BD Shifter*) to dynamically reduce the data to 4 bits for less precision degradation. For signed positive and negative operands, BD Shifter detects different leading bit. For positive operands, BD Shifter detects the first bit from the MSB that is equal to 1, marking it as leading "1". For negative operands, BD Shifter detects the first bit from the MSB that is equal to 0, marking it as leading "0". Different shift operations are performed depending on the leading bit position.

3.3 Sorting and Mode Selection for Sparsity Exploration

Sorting Module. To explore the model sparsity and determine the pruned part, we design a sorting module after the softmax function. By sorting the probabilities of different input tokens, we set the top-k maximize values as the important tokens and perform 8-bit computation. The other tokens are less important and can be calculated in a 4-bit format. The exploring method is based on the parallel sorting algorithm in FLiMS [12]. Specifically, our sorting module is based on the basic block called k-merger, as shown in the sorting module in Fig.1. Each k-merger merges two sorted lists of k elements to a sorted list of $2k$ elements in $logk$ steps. Another kind of merger can sort two elements and output the sorted list sequentially, which can be regarded as 1-merger but just outputs half of the sorted list at one time. This merger is called half-merger. In each step, k-merger performs $k/2$ compare-and-swap operations in parallel. Therefore, the hardware resource consumption of each k-merger is $k(logk+1)$. The parallelism means the number of sorted elements can be produced at one time, which can be adjusted by combination of k-mergers and half-mergers. We can further adjust the parallelism to achieve a balance between hardware consumption and throughput.

Mode Selection Module. The computation mode is controlled by related instructions generated from the compiler. Once the users only need a high-bit dense network after sparsity exploration, they can set the calculation mode to *dense*, otherwise the calculation mode can be set to *sparse*. While our *CSTrans-OPU* is under the *dense* mode, there is no need to perform low-bit computation with skipped sorting part. Therefore, the 4-bit PEs can be shifted and combined together to perform 8-bit multiplications, as shown in the mode selection module in Fig.1. In contrast, while our *CSTrans-OPU* is under the *sparse* mode, we use different precision of PEs for parts under diverse importance. In this way, no matter which mode and sparsity degree the model is under, we can perform matrix multiplication under a uniform format.

3.4 Post Process

The post-process module is composed of non-linear unit (NFU) and matrix add (MADD) unit. In the NFU, we implement the three types of non-linear functions in transformer networks, *i.e.*, softmax, layernorm and gelu according to the method in NN-LUT[19]. Specifically, we apply the piecewise linear approximation by a simple neuron network. In this way, the single input non-linear functions, *i.e.*, exponential, division, square root and gelu functions in above three non-linear functions are implemented by a 16-entry LUT.

3.5 Memory Management

Transformer networks always suffer in large model sizes and parameter numbers, leading to extremely massive memory footprint. Common storage space on FPGA chip is often not enough to accommodate full inference of these models. Therefore, the model parameters and information from CPU are divided into two groups. The first group is composed of parameters of larger scales, including input data along with trained weights and bias of linear layers. These parameters are sent to the off-chip DDR from CPU via PCIe interface and dynamically delivered to on-chip memory while inference. The second group consists of non-linear and quantization parameters. Since the sizes of them are small, they can be prewritten directly from CPU to on-chip memory.

4 COMPILER

To increase the flexibility and reduce the burden of deploying diverse transformer networks on our proposed *CSTrans-OPU* hardware architecture, we provide a full compilation process by a customized compiler. As shown in Fig.3, the compiler consists of five parts, *i.e.*, model parsing, operation fusion, model quantization, instruction generation and instruction reordering.

4.1 Model Parsing

Since users only get the model file of the target transformer networks, which can be rather different in model structure. Moreover, to express the transformer networks efficiently, various DL frameworks have been proposed nowadays, *e.g.*, TensorFlow, PyTorch, MXNet and Caffe. However, the model file of these DL frameworks are quite different. Therefore, we first need to parse the model information from the model file. In our work, we first convert model files under different DL frameworks into the unified format of ONNX and extract the structure and parameter details. We can also visualize the graph-level topology.

4.2 Operation Fusion

Although we have obtained the structure and parameters of the model, the operators of the transformer model are not supported by mainstream frameworks. In this way, the directly parsed structure is always fragmented and difficult to identify. Since our *CSTrans-OPU* supports the calculation of coarse-grained operators, *e.g.*, matrix multiplication, layernorm, gelu and softmax, operation fusion is essential to match the computation requirement. To fuse the operation, we introduce a keypoint-centric recursive algorithm as follows. The algorithm involves two main parts, *i.e.*, analyzing operator feature graphs and matching target graph structures.

Operator Feature Graph Analyzing. To leverage the characteristics of neural network operators, we employ the operator-centered graph algorithm. To fuse coarse-grained target operators or optimize target structures, we initially parse and extract features from their corresponding fine-grained operator graph structures. In the target graph structure, a key operator is selected as the center and the entire target structure is traversed in a step-wise radial outward direction to obtain a copy of the network structure's description information.

Target Graph Structure Matching. We perform the target sub-graph matching in the network structure. We first navigate the network model to identify all operators that match the key operators. Next, we expand the sub-neighbors with these operators at the center to divide a series of sub-domains. Within these sub-domains, we detect and match based on extracted descriptive information and successfully hit the target if it matches the target graph. We then identify corresponding modules in the sub-domains from the entire network. For the identified graph structure, we substitute it with a reconstructed coarse-grained operator to align the input and output ports, accomplishing the objective of operator fusion.

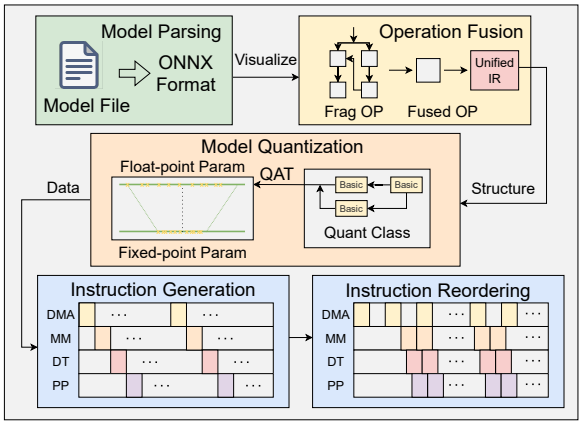

Figure 3: The component flows of our compiler.

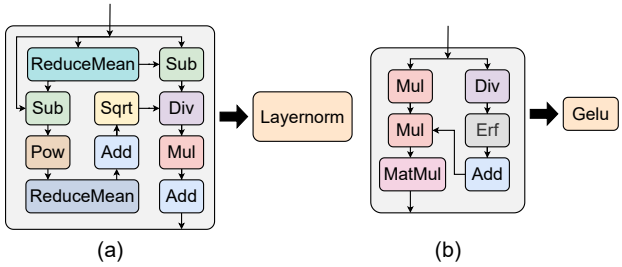

Figure 4: The comparison before and after operation fusion. (a) Layernorm function; (b) Gelu function.

In this way, the primary fragmented operations can be fused into our required granularity. Taking the layernorm and gelu functions as examples, Fig.4 shows the comparison before and after operation fusion. We can further transfer the fused graph into our defined intermediate representation (IR) for subsequent processing.

4.3 Model Quantization

To realize model quantization, we design two types of classes, *i.e.*, the basic class and the quantization class. The basic class includes the construction of all possible fused coarse-grained operators in transformer networks, which can be performed in both floating-point and quantization modes. By calling basic class in different modes, we can implement calculations for diverse demands.

We further construct the quantization class for our target model based on the basic class. Specifically, we identify the coarse-grained operations in our IR one by one, and call the related basic class according to the operation. Thus, the quantization class is a combination of successive calling to basic class.

Similarly, the quantization class contains both floating-point and quantization computations. After reading the IR, the contents of the basic class are added into the quantization class. Next, in the forward process, if the quantization class is set to floating-point mode,

the forward function returns the result of basic classifier. If the quantization class is set to quantization mode, it returns the classifier result and quantization loss. The quantization loss is the deviation between the original and quantize-aware training predictions. After the training converges, we can obtain the final quantized model, and the quantization scaling factors are transferred to our IR, which can be used for underlying hardware computation.

4.4 Instruction Generation

After the former optimization, we need to generate the executable instructions for our *CSTrans-OPU* hardware. The instruction set architecture follows the design concept of OPU [1, 17]. We customize our instructions into two types, *i.e.*, C-type and U-type instructions. C-type instructions contain the information of operation types and trigger condition, while U-type instructions are responsible for delivering related parameters to C-type instructions. The C-type instructions can be divided into four types:

• **DMA Control.** The DC instruction controls data transferring from DDR to BRAM.

• **Matrix Multiplication.** The MM instruction controls the computation pattern of the MMU. Note that different from original OPU instructions, there is a field of *mode* in our MM instruction. It determines either *dense* or *sparse* computation mode that our MMU performs.

• **Data Transfer.** The DT instruction is responsible for data transferring from FIFO array in our MMU to BRAMs.

• **Post Process.** The PP instruction controls the calculation of matrix addition and non-linear functions.

4.5 Instruction Reordering

Apart from operation fusion and network quantization, we also consider hardware-aware optimization in our compilation process, as shown in the part of instruction reordering in Fig.3. Specifically, we maximum the parallelism of our instruction by instruction reordering. In this way, multiple instructions can be performed in parallel to reduce execution time. We also minimize the number of BRAMs by releasing the used memory space in time. The related instructions of BRAM index and address are rearranged to realize the saving of BRAM usage.

5 EXPERIMENT

5.1 Experimental Setup

In this work, we implement our *CSTrans-OPU* on the Alveo U280 FPGA board. Additionally, we describe our *CSTrans-OPU* by Verilog HDL and synthesize it by Vivado 2020.2 tool chains while implementing the full compilation support in C++. Our experimental evaluation focuses on the compiler quantization performance, resource consumption, overlay performance analysis as well as comparison across different platforms.

5.2 Quantization Performance

According to the general quantization method in our compiler, we perform comparison experiments before and after quantization. Specifically, we conventionally compare the accuracy of 8-bit quantized network with baseline floating-point BERT-base network. The comparison results are shown in Table 1. Note that we choose the GLUE benchmark as the data set, in which spearman correlation is reported for STS-B, matthews correlation is reported for CoLA, the average of accuracy and f1 score is reported for MRPC and QQP, and the accuracy is reported for other tasks.

Table 1: The quantization performance of the compiler.

	MNLI	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE
wo quant	81.2	86.7	91.2	92.3	58.7	88.2	85.9	70.0
wi quant	**83.3**	**85.6**	**89.2**	**93.2**	**59.7**	**88.6**	**84.6**	**70.8**

Table 2: The resource utilization of *CSTrans-OPU*.

Resource	Usage	Available	Utilization(%)
LUT	821188	1303680	62.99
FF	848439	2607360	32.54
BRAM	1476	2016	73.21
DSP	3840	9024	42.55

5.3 Resource Utilization

In *CSTrans-OPU*, we use 2048 8-bit PEs and 1024 4-bit PEs in the multi-precision PE arrays. Due to the DSP-packing technology, both of them can perform 4096 multiplications in parallel for full resource utilization. We implement other computation modules mainly by LUTs, as the number of DSP is always the resource bottleneck on FPGA devices. The resource consumption details are shown in Table 2, where the target device we choose is Xilinx Alveo U280 FPGA with 9024 DSP slices, 70.88Mb BRAM and 1.30M LUTs in total.

5.4 Overlay Performance Analysis

As *CSTrans-OPU* is a reconfigurable overlay processor for diverse transformer networks, it is essential to demonstrate the flexibilty of the architecture. We test the inference latency on multiple transformer networks from various fields. Specifically, BERT-base, Roberta-base, ViT-base, DeiT-base, GPT-1 and Swin-T are considered for sufficient comparisons. The overlay performance of our *CSTrans-OPU* is shown in Table 3. As illustrated in the table, no matter which transformer network we implement, *CSTrans-OPU* shows better performance than other acceleration works.

5.5 Comparison Across Different Platforms

We also compare *CSTrans-OPU* with transformer network implementations in different platforms, and the results are shown in Table 4. For fair comparison, we test the same model of BERT-base under the sequence of 128 while unifying the data width to 8-bit integer. For comprehensive results of platforms at varied levels, we choose two target devices of different scales for both CPU and GPU comparisons. Specifically, for CPU experiments, we choose Intel(R) Core(TM) i7-8700 and Intel(R) Xeon(R) Gold 5218R. For GPU experiments, we choose NVIDIA K80 and RTX 3090. We also compare the performance with FPGA accelerators customized for fixed data flows. Both dense and sparse accelerator designs are considered to prove the advantages of our *CSTrans-OPU* under different scenarios. We set the token sparsity to 0.5 in our *CSTrans-OPU*, as it provides a trade-off between hardware performance and inference accuracy. Few FPGA-based processors have been proposed to achieve similar overlay support, we also separately compare with them for adequate evaluation. As shown in Table 4, *CSTrans-OPU* achieves 6.92-20.06× better throughput and 182.48× higher energy efficiency compared to the CPU applications. Compared with GPU platforms, we have 1.47-3.85× better overall throughput with 4.63-52.53× higher energy efficiency. In addition, we achieve up to 4.28× speedup and 4.94×

Table 3: Overlay inference performance of *CSTrans-OPU* compared with other FPGA designs.

Designs	FPGA	Model	Frequency(MHz)	Latency(ms)	DSP	BRAM	Norm.Power(mW/DSP)
CSTrans-OPU	**Xilinx Alveo U280**	**BERT-base/RoBERTa-base**	**200**	**7.23**	**3840**	**1476**	**2.29**
		ViT-base/DeiT-base		**11.96**			
		GPT1		**31.98**			
		Swin-T		**4.45**			
FTRANS [9]	Xilinx VCU118	RoBERTa-base	170	10.61	6531	/	3.85
NPE [8]	Zynq Z-7100	BERT-base	200	13.96	2020	369	9.90
ViTA [11]	Zynq ZC7020	ViT-base	150	363.64	220	35	4.00
VAQF [15]	Xilinx ZCU102	DeiT-base	150	40.32	1564	565	5.56
HPTA [7]	Xilinx ZCU102	BERT-base	200	6.09	2308	345	8.67
		Swin-T		6.72			
Vis-TOP [6]	Xilinx ZCU102	Swin-T	/	11.83	558	/	/

Table 4: Comparison of inference performance on different platforms.

	Plantform	Type	Latency (ms)	Throughput (GOPS)	Speedup Ratio	DSP	BRAM	Energy Efficiency (GOPS/W)	Efficiency Ratio
Intel(R) Core(TM) i7-8700	CPU	/	145.06	77.00	1.00	/	/	0.96	1.00
Intel(R) Xeon(R) Gold 5218R	CPU	/	50.00	223.40	2.90	/	/	/	/
NVIDIA K80	GPU	/	27.84	401.22	5.21	/	/	3.34	3.47
RTX 3090	GPU	/	10.61	1052.78	13.67	/	/	37.93	39.40
HAQ [10]	FPGA	accelerator	23.79	469.53	6.10	4272	1358	35.54	36.93
FTRANS [9]	FPGA	accelerator	10.61	1052.78	13.67	6531	/	42.01	43.65
SpAtten [16]	ASIC	accelerator	30.98	360.00	4.68	/	/	/	/
DTQAtten [18]	ASIC	accelerator	11.72	952.80	12.37	/	/	/	/
NPE [8]	FPGA	processor	13.96	800.14	10.39	2020	369	40.01	41.56
HPTA [7]	FPGA	processor	6.09	1833.44	23.82	2308	345	93.05	96.67
Vis-TOP [6]	FPGA	processor	11.83	726.37	12.26	558	/	/	/
CSTrans-OPU	**FPGA**	**processor**	**7.23**	**1545.45**	**20.06**	**3840**	**1476**	**175.64**	**182.48**

better energy efficiency over different accelerators. We are also up to 1.93× faster and 4.39× more energy efficient than other FPGA processors, suggesting that *CSTrans-OPU* is a promising solution for various transformer networks under different sparsity.

6 CONCLUSION

In this paper, we present *CSTrans-OPU*, an FPGA-based overlay processor with full compilation for transformer networks via sparsity exploration. In *CSTrans-OPU*, the architecture is reconfigurable and suitable to accelerate various transformer networks under different sparsity. A series of multi-precision PEs with DSP-packing are proposed to achieve unified computation of different sparsity with full resource utilization. The sorting and mode selection modules make it possible to explore network sparsity according to requirements of users. Moreover, we introduce a full compilation support by a straight-forward compiler for transformer networks. This compiler is responsible for parsing and optimizing network descriptions and generating dynamic and executable instruction streams. Evaluation results show that *CSTrans-OPU* is 6.92-20.06×, 1.47-3.85×, up to 4.28× and 1.93× faster and the energy efficiency is 182.48×, 4.63-52.53×, up to 4.94× and 4.39× better than CPU, GPU, mainstream customized accelerators and FPGA processors, respectively.

REFERENCES

[1] Bai et al. 2023. FET-OPU: A Flexible and Efficient FPGA-Based Overlay Processor for Transformer Networks. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*.

[2] Cao et al. 2022. Swin-Unet: Unet-like pure transformer for medical image segmentation. In *Proceedings of European Conference on Computer Vision (ECCV)*.

[3] Cao et al. 2023. PP-Transformer: Enable Efficient Deployment of Transformers Through Pattern Pruning. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*.

[4] Devlin et al. 2018. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[5] Dosovitskiy et al. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929* (2020).

[6] Hu et al. 2021. Vis-top: Visual transformer overlay processor. *arXiv preprint arXiv:2110.10957* (2021).

[7] Han et al. 2023. HPTA: A High Performance Transformer Accelerator Based on FPGA. In *2023 33rd International Conference on Field-Programmable Logic and Applications (FPL)*.

[8] Khan et al. 2021. NPE: An FPGA-based overlay processor for natural language processing. *arXiv preprint arXiv:2104.06535* (2021).

[9] Li et al. 2020. FTRANS: Energy-efficient acceleration of transformers using FPGA. In *Proceedings of ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED)*.

[10] Liu et al. 2021. Hardware acceleration of fully quantized bert for efficient natural language processing. In *Proceedings of IEEE Design, Automation & Test in Europe Conference & Exhibition (DATE)*.

[11] Nag et al. 2023. ViTA: A Vision Transformer Inference Accelerator for Edge Applications. *arXiv preprint arXiv:2302.09108* (2023).

[12] Papaphilippou et al. 2018. FLiMS: Fast lightweight merge sorter. In *2018 International Conference on Field-Programmable Technology (FPT)*.

[13] Peng et al. 2022. A length adaptive algorithm-hardware co-design of transformer on fpga through sparse attention and dynamic pipelining. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*.

[14] Radford et al. 2018. Improving language understanding by generative pre-training. *https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/language-unsupervised/language_understanding_paper.pdf* (2018).

[15] Sun et al. 2022. Vaqf: Fully automatic software-hardware co-design framework for low-bit vision transformer. *arXiv preprint arXiv:2201.06618* (2022).

[16] Wang et al. 2021. Spatten: Efficient sparse attention architecture with cascade token and head pruning. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*.

[17] Yu et al. 2019. OPU: An FPGA-based overlay processor for convolutional neural networks. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* (2019).

[18] Yang et al. 2022. DTQAtten: Leveraging dynamic token-based quantization for efficient attention architecture. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*.

[19] Yu et al. 2022. NN-LUT: Neural approximation of non-linear operations for efficient transformer inference. In *Proceedings of the 59th ACM/IEEE Design Automation Conference (DAC)*.