Mobile Accelerator Exploiting Sparsity of Multi-Heads, Lines, and Blocks in Transformers in Computer Vision

Eunji Kwon
Electrical Engineering
POSTECH
Pohang, Korea
eunjikwon@postech.ac.kr

Haena Song
Electrical Engineering
POSTECH
Pohang, Korea
hnsong@postech.ac.kr

Jihye Park
Electrical Engineering
POSTECH
Pohang, Korea
jihyepark@postech.ac.kr

Seokhyeong Kang
Electrical Engineering
POSTECH
Pohang, Korea
shkang@postech.ac.kr

***Abstract*—It is difficult to employ transformer models for computer vision in mobile devices due to their memory- and computation-intensive properties. Accordingly, there is ongoing research on various methods for compressing transformer models, such as pruning. However, general computing platforms such as central processing units (CPUs) and graphics processing units (GPUs) are not energy-efficient to accelerate the pruned model due to their structured sparsity. This paper proposes a low-power accelerator for transformers with various sizes of structured sparsity induced by pruning with different granularity. In this study, we can accelerate a transformer that has been pruned in a head-wise, line-wise, or block-wise manner. We developed a head scheduling algorithm to support head-wise skip operations and resolve the processing engine (PE) load imbalance problem caused by different number of operations in one head. Moreover, we implemented a sparse general matrix-to-matrix multiplication (sparse GEMM) module that supports line-wise and block-wise skipping. As a result, when compared with a mobile GPU and mobile CPU respectively, our proposed accelerator achieved 6.1× and 13.6× improvements in energy efficiency for the detection transformer (DETR) model and achieved approximately 2.6× and 7.9× improvements in the energy efficiency on average for the vision transformer (ViT) models.**

***Index Terms*—Energy-efficient Transformer Accelerator, Vision Transformer Optimization**

I. Introduction

Transformers have been actively utilized in computer vision for image classification, object detection, and semantic segmentation beyond the natural language processing (NLP) field. In object detection tasks, the detection transformer (DETR) [1] can achieve a higher mean average-precision (mAP) and frames-per-second (FPS), with only 41M parameters, than conventional model [2] based on convolutional neural networks (CNNs) with 166M parameters.

GPUs are commonly used for transformer computations, as most of their operations comprise matrix multiplications. There is a general misconception that transformer operations are specialized to GPU and that no more dedicated accelerators are required.

However, there are two main limitations that make general-purpose computing platforms insufficient for transformer acceleration. First, CPUs and GPUs simultaneously perform other tasks and deep learning applications. When their resources are shared by numerous background or foreground tasks, the CPU and GPU workloads increase, and the performance is significantly degraded. In mobile devices, most always-on deep learning tasks require constant performance, regardless of other tasks, rather than requiring excessively high levels of throughput.

Second, CPUs and GPUs are limited in their capability to accelerate optimized transformers with various sparsity. Conventional transformer's memory and computationally intensive properties do not readily allow for implementation on mobile devices due to their limited resources. Hence, various pruning strategies have been employed to lighten transformer models. However, reducing the number of parameters and operations does not necessarily lead to a proportional throughput. The unstructured or structured sparsity of the pruned model degrades parallelism. This makes it difficult to achieve the expected performance improvement.

To resolve these issues, dedicated hardware (HW) solutions were implemented with specialized pruning methods. Representative pruning techniques of transformers can be commonly classified into layer pruning [3], head pruning [4]–[7], line pruning (patch and dimension) [7]–[9], and block pruning [12] [13] depending on whether coarse-grained or fine-grained granularity is used. Previous studies demonstrated that such pruning methods can successfully reduce a large number of parameters and operations while minimizing the accuracy loss. For example, the vision transformer (ViT)-B16 model, which was pruned by 50% in heads and patches, demonstrated a reduction in FLOPs by 40.08%, 44.19%, and 42.08% based on cifar10, cifar100, and ImageNet, respectively, within an accuracy loss of 1% [7].

After pruning with different granularity (e.g., head, line, and block), the models exhibit sparsity with different sizes and in different regions. For example, line pruning generates the same number of consecutive zero weights as the row or column sizes of the weight matrices, whereas block pruning generates the same number of zero weights as the block size. This results in heterogeneous transformer layers [10], because each transformer layer exhibits a different number of heads in multi-head self-attention (MSA), and the number of operations of single-head attention (SHA) varied among heads. This reduces the utilization of processing engines (PE) and gradually leads to PE load-balancing issues.

In previous studies on pruning methodology, a reduction in the number of multiple parameters and FLOPs was achieved with a negligible loss of accuracy. However, limited research was conducted on methods to increase the speed and energy efficiency of transformers with limited computing resources. Motivated by this, we propose a head scheduling algorithm and mobile accelerator exploiting various sparsity: i) head-wise sparsity, ii) line-wise sparsity, and iii) block-wise sparsity. The main contributions of this study are as follows:

- We propose a head scheduling algorithm for two objectives: i) to skip unnecessary head operations, and ii) to resolve the load imbalance problem of SHA PEs that operate in parallel. The head scheduling algorithm linearly reduced the latency of the MSA in proportion to the head-wise pruning ratio. Without head-wise pruning, the latency was reduced by 10.21% when performing line-wise pruning by 70% on the ViT-Base due to the PE load balancing effect.
- We implement a mobile transformer accelerator that can skip the head-wise, line-wise, and block-wise operations. We propose a sparse GEMM to reduce latency by exploiting the row-wise sparsity of inputs, column-wise sparsity of weights, and block-wise sparsity of both of them. Our sparse GEMM can reduce the latency of matrix multiplication by 58.95% when the line-wise sparsity is 75% compared to FFT-IFFT PE used in [12]).

This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education (2022R1A6A3A13063601) and was supported by National R&D Program through the National Research Foundation of Korea (NRF) funded by Ministry of Science and ICT (2022M3H4A1A04096496).

978-3-9819263-7-8/DATE23/© 2023 EDAA

TABLE I: Qualitative comparison with previous studies on transformer Accelerators.

	A^3 [11] (HPCA 2020)	FTRANS [12] (ISLPED 2020)	SpAtten [4] (HPCA 2021)	SALO [14] (DAC 2022)	Proposed Accelerator
Application Field	NLP	NLP	NLP	NLP, Vision	Vision
Transformer Optimization Method	Unstructured Sparsity	Block-Circulant Matrix (BCM)-based Compression	Coarse-grained Structured Sparsity (Head, Token)	Sparse Attention Mechanism (Sparse Window / Dilated Window)	Coarse- and Fine-grained Structured Sparsity (Head, Line, Block)
Specialty of SW/HW Implementation	Greedy Approximate Attention	FFT-IFFT PE	Algorithm-Architecture Codesign	Data Scheduler (Data Splitting / Reordering)	Head Scheduling Algorithm, Sparse GEMM PE
Hardware Platform	ASIC / Verilog	FPGA (VCU 118) Vivado HLS	ASIC / Verilog	ASIC/ Chisel	FPGA (ZCU 104) Vivado HLS

- We analyze the trade-off between accuracy and various pruning ratios (head and line) of the transformers: i) DETR [1] for object detection and ii) ViT [15] [16] for image classification. As a result, accuracy loss was minimized by pruning with different granularity, which means that HW is needed to support various sparsity. Using the proposed accelerator, the energy efficiency of the DETR is 6.1× and 13.6× higher than mobile GPU and CPU within 3.7 average precision (AP_{50}) drop. The average energy efficiencies of the ViT models (ViT-Tiny, ViT-Small, and ViT-Base) were 2.6× and 7.9× higher than mobile GPU and CPU within a 2% accuracy loss.

The remainder of this paper is organized as follows: Section II reviews related studies and Section III presents the proposed head-wise, line-wise, and block-wise skipping methods. Section IV presents the overall architecture of the proposed accelerator, which includes SHA PEs. Section V presents the experimental setup and results and Section VI concludes this paper.

II. Preliminary

A. Background of Transformer

The transformer encoder comprises multiple encoder layers stacked in series [17]. The encoder layers consist of MSA and a feed-forward neural network (FFN). The decoder has multiple decoder layers consisting of MSA, multi-head cross attention (MCA), and FFN. The multi-head attention (MHA) (Eq. (1)) including MSA and MCA, is the result of concatenating the total number of *H* single-head attentions (SHA) (Eq. (2)).

$$MultiHead(Q,K,V) = Concat(head_1, ..., head_H)W_O \quad (1)$$

$$head_i = Attention(QW_Q^i, KW_K^i, VW_V^i) \quad (2)$$

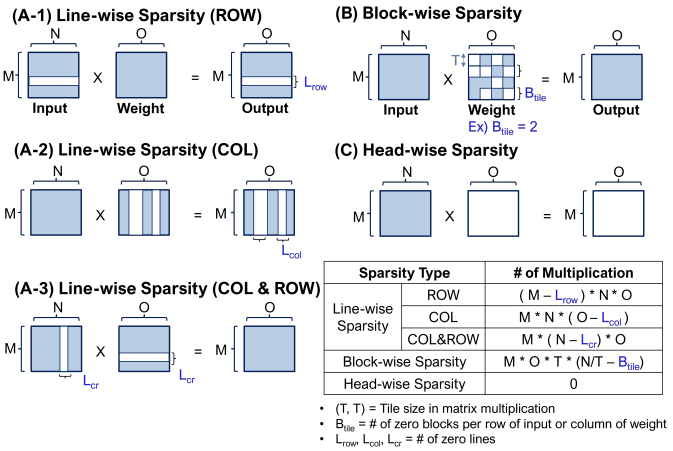
$$Attention(Q_i,K_i,V_i) = Softmax(Q_iK_i^T/\sqrt{d_k})V_i \quad (3)$$

In SHA, the query vector (Q_i), key vector (K_i), and value vector (V_i) are generated by multiplying the input query (Q), key (K), value (V), and trainable weights W_Q^i, W_K^i, and W_V^i, respectively. After embedding, scaled-dot product attention (SDP attention) is performed to determine the extent to which a given feature is closely related to other features (Eq. (3)).

In this paper, the representative object detection (DETR [1]) and image classification (ViT [15] [16]) models were selected and tested to demonstrate the generality of the proposed method. DETR used both the transformer encoder and decoder structures. The decoder layers of the model received several object queries as initial inputs and utilized the encoded information. In ViT, the image was cut into patches and is utilized as the input of the encoder of the transformer, which is similar to creating an image in the form of a word sequence.

B. Transformer Accelerator

Recently, several types of research have been conducted to accelerate optimized transformers. Accordingly, transformer accelerators can be classified based on the optimization method of transformer models. TABLE I represents a qualitative comparison with previous studies on transformer accelerators. Ham et al. [11] approximated attention mechanisms by selecting important key elements to solve the computational cost of the attention, but the accelerator used unstructured sparsity and required additional logic for computing the important candidate selection. Li et al. [12] reduced memory usage by compressing the model parameters into a block-circulant matrix (BCM) and proposed FTRANS accelerator to reduce computations by replacing general matrix multiplication operations with fast Fourier transform (FFT) operations. However, they applied BCM compression only to FFN. Moreover, it is difficult to use on mobile devices due to its high power consumption. Wang et al. [4] proposed a software-hardware (SW-HW) co-design framework for pruned transformers. Wang et al. [4] focused on the compression techniques for the cascade head and token pruning of a transformer and its HW accelerator, SpAtten. However, they only performed coarse-grained pruning of the transformer models, so their sparsity ratio was relatively low. Shen et al. [14] proposed an accelerator for sparse attention mechanisms, such as sparse-window attention and dilated-window attention, to solve the problem that the computation amount of the attention mechanism exhibits a quadratic with the sequence length. However, it is difficult to apply this approach to vanilla attention mechanisms, as the hardware is specific to several sparse attention models.



Sparsity Type		# of Multiplication
Line-wise Sparsity	ROW	(M – L_{row}) * N * O
	COL	M * N * (O – L_{col})
	COL&ROW	M * (N – L_{cr}) * O
Block-wise Sparsity		M * O * T * (N/T – B_{tile})
Head-wise Sparsity		0

- (T, T) = Tile size in matrix multiplication
- B_{tile} = # of zero blocks per row of input or column of weight
- L_{row}, L_{col}, L_{cr} = # of zero lines

Fig. 1: Various types of sparsity in transformer: (Type A) line-wise sparsity, (Type B) block-wise sparsity, and (Type C) head-wise sparsity.

C. Sparsity Types Exploited by Proposed Method

Fig. 1 presents the different types of the sparsity of the transformer model in the proposed accelerator. We defined three types of sparsity in this paper: line-wise, block-wise, and head-wise sparsity. Patch or token pruning leads to row-wise sparsity of the inputs (Type A-1).

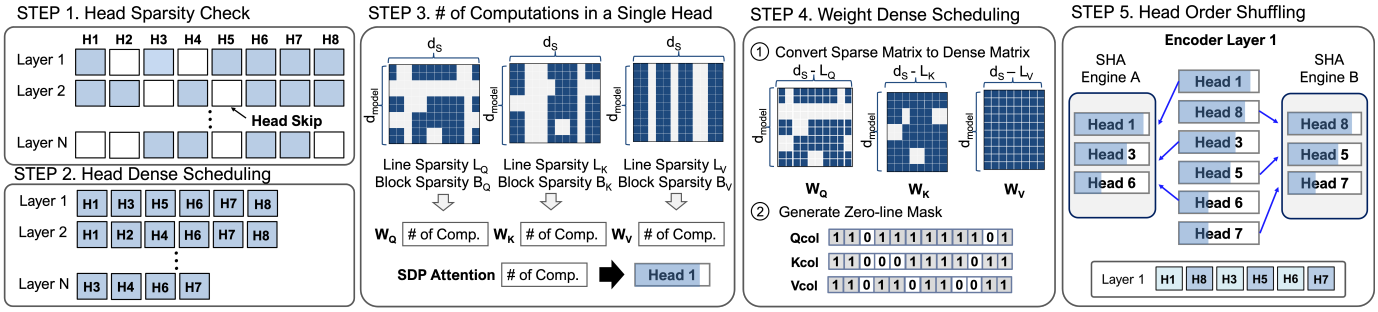


Fig. 2: Head scheduling algorithm for two objectives: i) to skip unnecessary head operations (STEPS 1-2) and ii) to resolve the load imbalance problem of several SHA PEs operating in parallel (STEPS 3-5).

Column or dimension pruning of SHA leads to column-wise sparsity of the weights (Type A-2). Dimension pruning of the transformer model leads to column-wise sparsity of the inputs and row-wise sparsity of the weights (Type A-3). Block pruning induces block-wise sparsity (Type B). Moreover, zero inputs generated through SoftMax or ReLU activation function in the transformer operation can be clustered to form block-wise sparsity. Head pruning induces head-wise sparsity (Type C), i.e., all weights constituting a single head in MSA are zero. FTRANS [12] accelerator exploits Type B sparsity and SpAtten [4] architecture exploits Type A-1 and C sparsity. The proposed accelerator exploits all types of sparsity in transformers.

The proposed accelerator can reduce the number of operations in matrix multiplication (input $[M \times N]$ and weight $[N \times O]$) by fully exploiting sparsity, as shown in Fig. 1. Without sparsity, the number of multiplications is $M \cdot N \cdot O$. With three types of line-wise sparsity, we can skip the number of multiplications by $L_{row} \cdot N \cdot O$, $L_{col} \cdot M \cdot N$, and $L_{cr} \cdot M \cdot O$. With block-wise sparsity, we can skip the number of multiplications by $M \cdot O \cdot T \cdot B_{tile}$, where T is the matrix tiling size and B_{tile} is the number of zero blocks in the operations required for one output tile matrix. With head-wise sparsity, the entire matrix multiplication can be skipped.

III. Proposed Skipping Methods

A. Head-wise Skipping and Head Shuffling

When computing resources are sufficient such as GPUs, MHA can be calculated simultaneously. On the other hand, all multi-heads cannot be operated in parallel in resource-constrained devices. Therefore, we complete the MHA operations by placing several SHA PEs and by reusing the engines and concatenating the results of SHA; it would improve the latency as the head-wise pruning ratio increases, unlike GPUs.

In addition, the remaining heads after head-wise pruning have different workloads due to additional line and block pruning. The problem of an unbalanced workload occurs in the SHA when the PEs operate in parallel. Therefore, we propose a new dataflow that operates SHA by mixing the head order and grouping the remaining heads with a similar number of operations. A simple but effective proposed algorithm is essential to operate transformers in edge devices efficiently.

Fig. 2 presents the head scheduling algorithm for two objectives: i) to skip unnecessary head operations (Steps (1)-(2)) and ii) to solve the load imbalance problem with two SHA processing engines operating in parallel (Steps (3)-(5)). In Step (1), the pruned heads in all layers are identified, and the zero weights constituting a single-head are removed, such that unnecessary head operations can be skipped. In Step (2), the remaining weights stored behind the pruned heads are pulled for dense weight scheduling. In Step (3), we check the row-wise and column-wise sparsity of the matrices. Thereafter, we obtain the number of operations required per head in one layer by utilizing line-wise and block-wise sparsity. We consider the table in Fig. 1 when computing the computational overhead. In Step (4), sparse matrices are converted into dense matrices by removing all the zero lines that do not require operations to save memory storage. Moreover, we generate a zero-line mask that contains whether each line is to be skipped. In Step (5), we shuffle the order of the head operations by grouping two heads with similar operations among the heads in one encoder layer. More precisely, when sorting two pairs in descending order by the number of operations, the paired heads are distributed among the two SHA PEs.

TABLE II: Parameter descriptions in transformer models.

Symbol Name	Description of the Parameters in Transformer
H	The number of heads in multi-head attention
S, S_D	Input sequence length of the transformer's encoder and decoder
d_{model}	Embedding dimension of the transformer model
d_s	Embedding dimension of single-head attention, d_s = d_{model} / H
d_{ffn}	Hidden dimension of feed-forward neural network

A head scheduling index is generated using the abovementioned head scheduling algorithm. When processing the MHA, the HW is configured to skip the computation of SHA by storing all elements of the attention value as zero if the head number is not included in the head scheduling index (e.g., *H2* and *H4*). The HW executes SHA in order using two SHA PEs in accordance with shuffled head operations and stores them in the appropriate memory location.

B. Line-wise Skipping and Block-wise Skipping

We propose the sparse general matrix-to-matrix multiplication (sparse GEMM) as shown in Fig. 3 to implement line-wise and block-wise skipping methods. When implementing the sparse GEMM that multiplies matrix *A* $[M \times N]$ and matrix *B* $[N \times O]$, and exports matrix *C* $[M \times O]$, we check the row-wise sparsity of matrix *A* and column-wise sparsity of matrix *B*. Given that, all elements in the i^{th} row of the input matrix *A* are zero, then all elements in the i^{th} row of the output matrix *C* are zeros. If the j^{th} column of the input matrix *B* is zero-line, then the j^{th} column of the output matrix *C* is zero-line.

We generate the dense-scheduled data and zero-line mask by using the proposed head scheduling algorithm and put them as the inputs of the proposed sparse GEMM. The sparse GEMM performs matrix multiplication with the obtained dense matrices A' $[(M\text{-}\alpha) \times N]$ and B' $[N \times (O\text{-}\beta)]$ based on tiling. In exceptions where the size of the input matrix is not a multiple of the tiling size, the module performs tiled matrix multiplication at the end by filling the remainder of the tiled matrix with zero values (e.g. A_4 and A_8 in Fig. 3-2), and does not store them in the output matrix. To apply block-wise skipping, the proposed HW is implemented to verify in real time whether the tiled matrix is the zero tile and skip the multiplications of the tiled matrices that contain a zero tile.

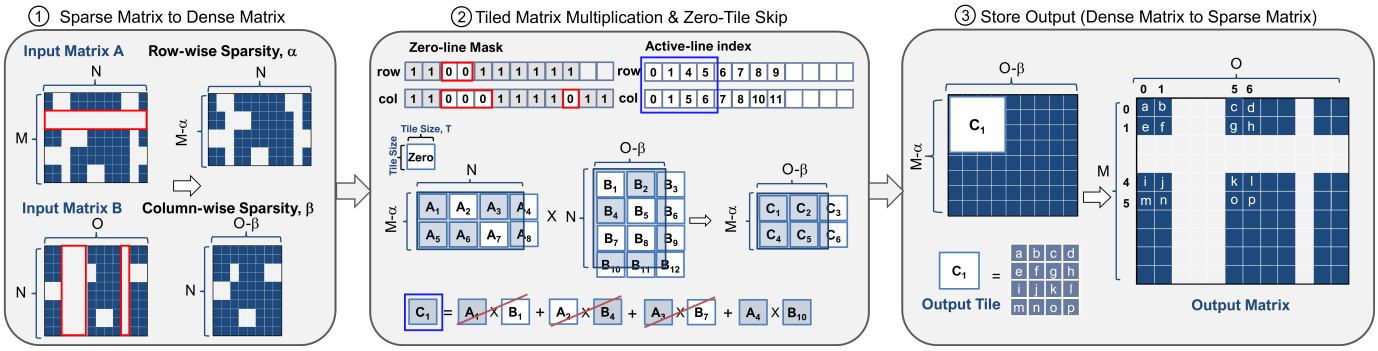


Fig. 3: Explanation of the proposed GEMM module. Row-wise or column-wise sparse matrices are converted into dense matrices in advance to support line-wise skipping. The HW refers to index bits equal to the row and column sizes, to store which lines are non-zero lines. The GEMM module performs matrix multiplication based on tiling strategy. The proposed module can skip matrix multiplications, which include zero-tile to support block-wise skipping.

TABLE III: FPGA resource utilization, power and throughput reports of the accelerators for DETR, ViT-Tiny, ViT-Small, and ViT-Base. Except for the ViT-Tiny model, two SHA PEs were placed in the proposed accelerator.

Resource Utilization	**Available Resource**	DETR 185MHz	ViT-T 185MHz	ViT-S 100MHz	ViT-B 100MHz
BRAM18K [%]	624	74	80	58	93
DSP [%]	1728	25	23	19	19
FF [%]	460800	18	24	16	16
LUT [%]	230400	69	87	31	64
LUTRAM [%]	96	33	55	61	36
# SHA PE / # Head		2 / 8	3 / 3	2 / 6	2 / 12
Total Power [W]		6.09	7.33	4.66	4.62
PL Power [W]		3.35	4.56	1.92	1.88
Throughput [GOP/s]		36.1	12.0	16.2	20.9

Finally, the dense-matrix is converted to sparse-matrix to complete the operation, which returns the zero data excluded from the operation to the appropriate locations. However, the aforementioned post-processing requires buffer of the same size as the dense output matrix C' [$(M\text{-}\alpha)\times(O\text{-}\beta)$]. Instead, we calculate the output index in advance and immediately store the output in appropriate locations without being stored in the buffer to eliminate the buffer overhead required for the dense matrix. In particular, patch pruning in the input image can increase row-wise sparsity, and column pruning of weights can increase column-wise sparsity; when using the sparse GEMM, the latency improvement increases as the sparsity increases.

IV. Hardware Implementation

A. Single-Head Attention (SHA) Processing Engine (PE)

In Fig. 4, the GEMM outputs the result of the matrix multiplication of matrix A and matrix B, $(A\times B)$, whereas SDP attention outputs the result of the scaled-dot product attention, $(A\times B^T/C)$. In particular, SDP attention performs the matrix multiplication of matrix A and matrix B^T directly without transposing matrix B to reduce the overhead of the matrix transposition. In the proposed GEMM and SDP attention modules, the parts that load data into the tile buffer, the computation part, and the part that stores the final output tile in the memory are fully-pipelined. We designed SoftMax modules to reduce the computational overhead by transforming the SoftMax equation as follows [18]:

$$SoftMax(x_i) = exp(x_i - x_{max} - ln(\sum_{j=1}^{N} exp(x_j - x_{max}))) \quad (4)$$

B. Dataflow of the Proposed Accelerator

Fig. 4 illustrates the dataflow of the encoder layer in ViT-Base as an example. We used the double buffering method in all steps in which the HW module processes the data of the one buffer, whereas the input channel stores the data in the other buffer, which is required for following the HW module. All input and weight data are loaded through AXI direct memory access (DMA) without passing through the processing system (PS).

STEPS (1-4). In MSA, the concatenated matrix is generated by performing the same number of SHA operations as the total number of remaining heads (H_r = H - # of pruned heads). If there is no single-head index in the head scheduling index, then the operation of this SHA is skipped and the attention value, which is the output of the SHA, is filled with zero instead. Based on the order in which the heads are shuffled for each layer, the operation is performed using two SHA PEs in the appropriate order, and obtained two attention values are stored in an proper location.

In the SHA PE, the embedding vector of the query (Q_i), key (K_i), and value (V_i) are generated by performing matrix multiplications of the input query (Q), input key (K), and input value (V), respectively, with each weight (W_Q, W_K, W_V) in Steps (A-D). In Steps (D-F), the SDP attention performs scaled-dot product attention (Eq. (3)) to generate attention score matrix QK_i. The GEMM outputs attention value A_i by performing matrix multiplication of attention distribution, which is the output after SoftMax, and value vector V_i. Consequently, two attention values are created in parallel because there are two SHA PEs in the accelerator. The accelerator repeats Steps (A-F) four times (H_r / # of SHA PEs).

STEP (5). Thereafter, the GEMM module performs matrix multiplication between the concatenated matrix, A_O, and the weight matrix W_O to complete the multi-head attention matrix.

STEPS (6-9). Two additional GEMM modules are used in FFN. When the operations of MSA and FFN are completed, the Add&Norm module adds the previous input to the output and normalizes it.

V. Experiments

A. Experimental Setup

1) Pruning Method: We conducted an experiment using DETR and ViT models to demonstrate that the latency can be decreased while minimizing accuracy loss after pruning with different granularity (head and line). In head pruning, the number of heads was even for each layer after pruning, given that two SHA PEs were placed in the proposed transformer accelerator. In addition, the column weights in MSA and the row and column weights in FFN were pruned depending on the importance of the weights using importance scores [19]. In the ViT-Tiny, only line pruning of MSA and FFN was applied because the model has three heads.

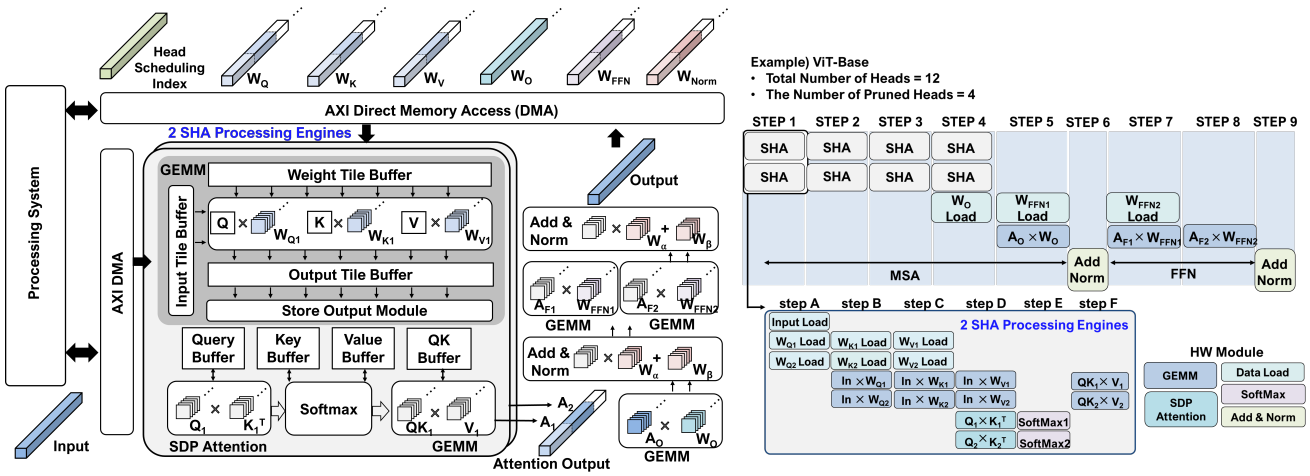


Fig. 4: Dataflow of transformer engine in the proposed accelerator.

2) Synthesis Report of the Proposed Accelerators: TABLE III lists the synthesis results of the proposed HW in *Vivado* high-level synthesis (HLS). We synthesized the HW for the DETR and ViT models on a *Xilinx ZCU104* FPGA board. The HW accelerators for the DETR and ViT-Tiny were synthesized at 185MHz to determine achievable latency improvement for the similar amount of power consumption of the mobile GPU and CPU; Mobile GPU (*NVIDIA Jetson Nano*) and CPU (*ARM Cortex A57*) consumed an average of 7W when running transformer models. The power consumption represented by the PS was approximately 2.7W; thus, the proposed programmable logic (PL) is appropriate for implementation on low-power embedded devices.

B. Experimental Results of Proposed Methods

1) Results of Head Scheduling Algorithm: The head scheduling algorithm reduced the latency in two parts: 1) head-wise skipping and 2) head shuffling. Latency reduction in MSA due to head-wise skipping was linearly reduced in proportion to the number of pruned heads. Additionally, head shuffling was found to reduce latency by 10.2% on average with 70% line pruning of the ViT-Base (Fig. 5-a).

2) Comparison with FTRANS: The TABLE IV shows a comparison between our proposed accelerator and the FTRANS accelerator [12]. Compared to the FTRANS, the power efficiency of the proposed one was improved by 55.8%, and the throughput per DSP was improved by 3.2 times. It was complex to conduct a direct comparison between the proposed and the previously developed hardware, given that the target FPGA boards and transformer models were different. Therefore, we compared the latency between the sparse GEMM with the line pruning and FFT-IFFT PE with the BCM compression method used by FTRANS. We conducted an experiment to confirm the decrease in the latency of the matrix multiplication with respect to the line-wise sparsity ratio (Fig. 5-b). FTRANS applied BCM compression with the block size of 8, which indicates that the number of parameters in the weights was compressed into 1/8, only to the FFN whereas we applied column pruning by 75% to the FFN. As a result, the latency of the proposed GEMM decreased by 58.95% when compared with the FFT-IFFT PE.

C. End-to-End Results of Latency and Energy Efficiency

Fig. 6 represents the results of the AP and average recall (AR) of the DETR and the accuracy results of the ViT-Base for various head and line pruning ratios. The accuracy drop was nearly negligible in line pruning, whereas it gradually increased as head pruning ratio

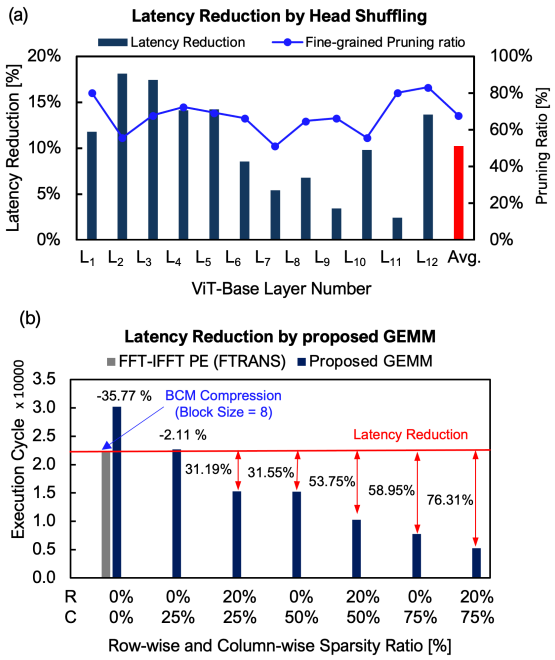


Fig. 5: (a) Latency reduction by head shuffling. (b) Latency comparison between the proposed GEMM and FFT-IFFT PE in FTRANS [12].

TABLE IV: Comparison between the proposed accelerator and FTRANS [12].

	Proposed Accelerator	**FTRANS [12]**
Target Model	DETR (Vision)	BERT (NLP)
Throughput [GOP/s]	36	170
PL Power [W]	3.4	25
Power Efficiency [GOP/s·W]	10.6	6.8
Throughput / DSP	0.083	0.026
Throughput / kLUT	0.226	0.377

increased. Even if the same multi-head dimension remained after head and line pruning in MSA, the accuracy drop would be smaller for a higher number of heads, such that SHA would be further performed. Based on the experimental results, it is necessary to reduce more parameters by pruning with different granularity. This proves the need for the proposed HW to be capable of supporting various sparsity.

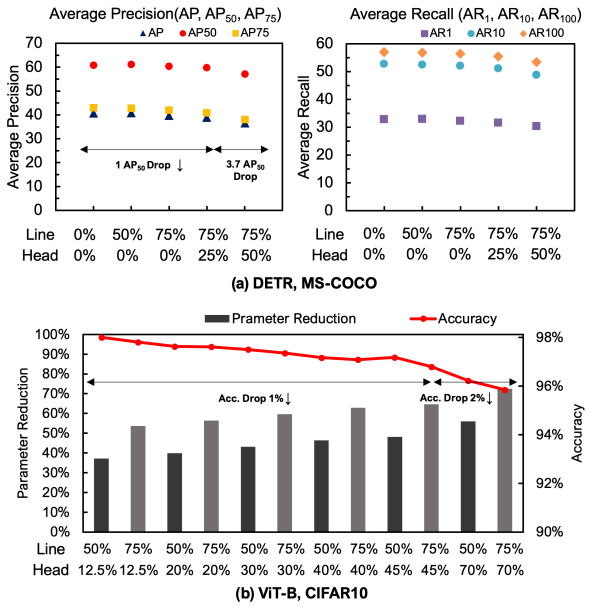


Fig. 6: (a) The results of average precision (AP) and average recall (AR) of the DETR with respect to different pruning ratios. n refers to the number of object queries in AR_n. (b) The trade-off between the accuracy and the reduction in the number of parameters of the ViT-Base with respect to different pruning ratios (head pruning in MSA and line pruning in MSA and FFN.)

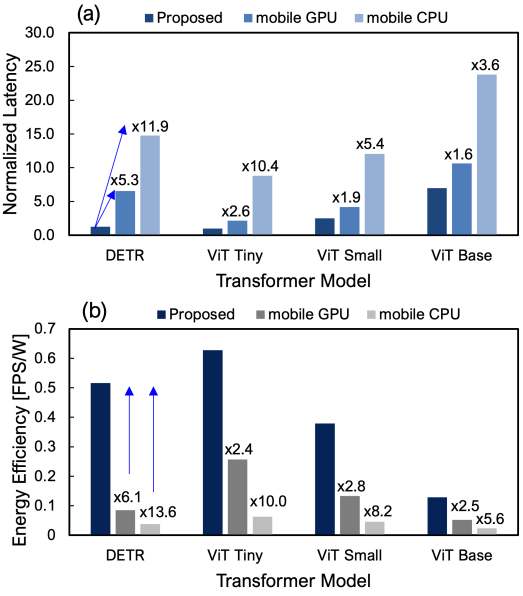


Fig. 7: (a) Normalized latency and (b) energy efficiency (FPS/total power [FPS/W]) comparison of the DETR, ViT-Tiny, ViT-Small, and ViT-Base with general computing platforms, namely, a mobile GPU (NVIDIA Jetson Nano) and mobile CPU (ARM Cortex A57 quad cores.)

We measured the latency of the DETR based on 50% head and 75% line pruning ratios and the latency of the ViT-Small and ViT-Base based on 70% head and 75% line pruning ratios; we applied only line pruning on the ViT-Tiny. In the case of the DETR, the proposed HW was nearly 5.3×, 11.9× faster than the mobile GPU and CPU, respectively, while maintaining less than five AP_{50} drops. In the case of the ViT-Tiny, ViT-Small, and ViT-Base, the proposed accelerator achieved approximately 2.0× and 6.5× on average faster than the GPU and CPU, respectively, while maintaining the accuracy loss of the three models within 2%. Additionally, the energy efficiency of the accelerator was 6.1× and 13.6× higher in DETR, and the average energy efficiency of the accelerator for ViT models was 2.6× and 7.9× higher in the GPU and CPU, respectively.

VI. Conclusion

After pruning, transformers exhibit various types of sparsity: i) head-wise sparsity, ii) line-wise sparsity, and iii) block-wise sparsity. This paper proposes a general-purpose accelerator that can skip the head-wise, line-wise, and block-wise operations of the model. First, a head scheduling algorithm is proposed for head-wise skipping. Second, sparse GEMM is proposed for line-wise and block-wise skipping. The energy efficiency of transformer models was improved with heterogeneous dimensions per layer or sparse structures created by pruning with different granularity. Consequently, the proposed accelerator demonstrated a superior improvement in latency and energy-efficiency when compared with general computing platforms.

References

[1] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers", *European conference on computer vision (ECCV)*, 2020, pp. 213-229.

[2] S. Ren, K. He, R. Girshick, and J. Sun, " Faster r-cnn: Towards real-time object detection with region proposal networks", *Advances in neural information processing systems (NeurIPS)*, 2015, 28.

[3] A. Fan, E. Grave, and A. Joulin, "Reducing transformer depth on demand with structured dropout", *arXiv preprint arXiv:1909.11556*, 2019.

[4] H. Wang, Z. Zhang, and S. Han, "Spatten: Efficient sparse attention architecture with cascade token and head pruning", *IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2021, pp. 97-110.

[5] E. Voita, D. Talbot, F. Moiseev, R. Sennrich, and I. Titov, "Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned", *arXiv preprint arXiv:1905.09418*, 2019.

[6] P. Michel, O. Levy, and G. Neubig, "Are sixteen heads really better than one?", *Advances in neural information processing systems (NeurIPS)*, 2019, 32.

[7] Z. Song, Y. Xu, Z. He, L. Jiang, N. Jing, and X.Liang, "CP-ViT: Cascade Vision Transformer Pruning via Progressive Sparsity Prediction", *arXiv preprint arXiv:2203.04570*, 2022.

[8] J. Mao, H. Yang, A. Li, H. Li, and Y. Chen, "Tprune: Efficient transformer pruning for mobile devices", *ACM Transactions on Cyber-Physical Systems*, 2021, 5(3), pp. 1-22.

[9] K. Murray, J. Kinnison, T. Q. Nguyen, W. Scheirer, and D. Chiang, "Auto-sizing the transformer network: Improving speed, efficiency, and performance for low-resource machine translation", *arXiv preprint arXiv:1910.06717*, 2019.

[10] H. Wang, Z. Wu, Z. Liu, H. Cai, L. Zhu, C. Gan, and S. Han, " Hat: Hardware-aware transformers for efficient natural language processing", *arXiv preprint arXiv:2005.1418*, 2020.

[11] T. J. Ham, S.J. Jung, S. Kim, Y. H. Oh, Y. Park, Y. Song and D. K. Jeong, "A^3: Accelerating attention mechanisms in neural networks with approximation", *IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2020, pp.328-341.

[12] B. Li, S. Pandey, H. Fang, Y. Lyv, J. Li, J. Chen, and C. Ding et al., "Ftrans: energy-efficient acceleration of transformers using fpga", *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED)*, 2020, pp. 175-180.

[13] H. Peng, S. Huang, T. Geng, A. Li, W. Jiang, H. Liu, and C. Ding et al., "Accelerating transformer-based deep learning models on fpgas using column balanced block pruning", *International Symposium on Quality Electronic Design (ISQED)*, 2021, pp. 142-148.

[14] G. Shen, J. Zhao, Q. Chen, J. Leng, C. Li, and M. Guo, "SALO: an efficient spatial accelerator enabling hybrid sparse attention mechanisms for long sequences", *Design Automation Conference (DAC)*, 2022, pp. 571-576.

[15] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, T., and N. Houlsby et al., "An image is worth 16x16 words: Transformers for image recognition at scale", *arXiv preprint arXiv:2010.11929*, 2020.

[16] A. Steiner, A. Kolesnikov, X. Zhai, R. Wightman, J. Uszkoriet, and L. Beyer, "How to train your vit? data, augmentation, and regularization in vision transformers", *arXiv preprint arXiv:2016.10270*, 2021.

[17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, and I. Polosukhin et al., "Attention is all you need", *Advances in neural information processing systems (NeurIPS)*, 2017, 30.

[18] 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", *International System-on Chip Conference (SOCC)*, 2020, pp. 84-89.

[19] M. Zhu, Y. Tang, and K. Han, "Vision transformer pruning", *arXiv preprint arXiv:2104.08500*, 2021.

[20] L. Lu, Y. Jin, H. Bi, Z. Luo, P. Li, T. Wang and Y. Liang, "Sanger: A co-design framework for enabling sparse attention using reconfigurable architecture.", *International Symposium on Microarchitecture.*, 2021, pp. 977-991.