Zebra: Leveraging Diagonal Attention Pattern for Vision Transformer Accelerator

Sukhyun Han*, Seongwook Kim†, Gwangeun Byeon†, Jihun Yoon*, and Seokin Hong†
*Department of Semiconductor Convergence Engineering, †Department of Electrical and Computer Engineering
Sungkyunkwan University, Suwon, Republic of Korea
{kavin1010, su8939, kebyun, head06, seokin}@skku.edu

***Abstract*—Vision Transformers (ViTs) have achieved remarkable performance in computer vision, but their computational complexity and challenges in optimizing memory bandwidth limit hardware acceleration. A major bottleneck lies in the self-attention mechanism, which leads to excessive data movement and unnecessary computations despite high input sparsity and low computational demands. To address this challenge, existing transformer accelerators have leveraged sparsity in attention maps. However, their performance gains are limited due to low hardware utilization caused by the irregular distribution of non-zero values in the sparse attention maps.**

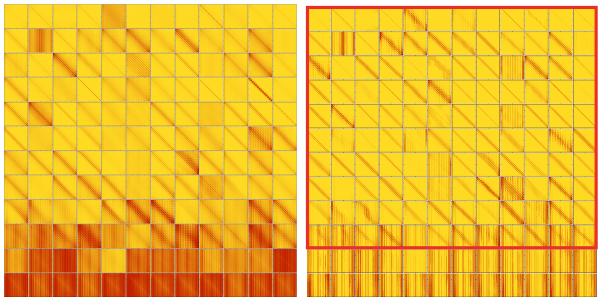
Self-attention often exhibits strong diagonal patterns in the attention map, as the diagonal elements tend to have higher values than others. To exploit this, we introduce *Zebra*, a hardware accelerator framework optimized for *diagonal attention patterns*. A core component of Zebra is the *Striped Diagonal (SD) pruning* technique, which prunes the attention map by preserving only the diagonal elements at runtime. This reduces computational load without requiring offline pre-computation or causing significant accuracy loss. Zebra features a *reconfigurable accelerator architecture* that supports optimized matrix multiplication method, called *Striped Diagonal Matrix Multiplication (SDMM)*, which computes only the diagonal elements of matrices. With this novel method, Zebra addresses low hardware utilization, a key barrier to leveraging the diagonal patterns. Experimental results demonstrate that Zebra achieves a 57× speedup over a CPU and 1.7× over the state-of-the-art ViT accelerator with similar inference accuracy.

***Index Terms*—Accelerator architectures, Attention mechanisms, Computer vision, Sparse matrices**

I. Introduction

Transformer models have recently demonstrated outstanding performance across various domains [1]–[3]. Among them, Vision Transformers (ViTs) have shown excellent capabilities in computer vision, a field that has been primarily driven by convolution neural networks (CNNs) [4]–[8]. In ViTs, the self-attention (SA) mechanism enables the model to capture relationships between patch tokens in the input image. Even if the SA mechanism is a core operation for high inference accuracy of ViTs, it has quadratic complexity with respect to the number of tokens, leading to high latency in the inference tasks [9], [10]. Additionally, this mechanism involves significant data movement and unnecessary computations, making it difficult to accelerate ViTs on hardware platforms despite their inherent sparsity in the attention matrix and relatively low computational demands compared to other operations [11].

Several studies have proposed utilizing the sparsity to reduce the computational complexity based on the observation that most tokens are weakly connected to others in ViTs [12]–[15]. Researchers have proposed software and hardware co-design approaches to remove unnecessary tokens and heads [12], [15] or to exploit sparse attention patterns [9], [13], [14]. However, these approaches often encounter limited performance improvement due to significant prediction overhead in identifying unimportant information and low sparsity levels that can be achieved. A recent prior work [13] prunes and polarizes the attention map to have denser or sparser fixed patterns, significantly reducing computation without hurting accuracy. They also propose an auto-encoder module that effectively reduces high-cost data movement from memory. However, the random distribution of non-zero values resulting from pruning still leads to low hardware utilization, hindering parallel processing on hardware.



(a) Original score maps (b) Pruned score maps

Fig. 1: Visualization of attention score maps in DeiT-Base [16].

In ViTs, a consistent diagonal pattern resembling zebra stripes can be observed in the attention score map since the information from the surroundings of each input patch token is most crucial [9], [17]. Figure 1 shows original attention score maps and score maps pruned to an 80% sparsity level using the magnitude-based pruning method. In these score maps, the elements shown in red represent larger values, while those in yellow indicate values close to zero. As highlighted by the red box in the figure, a clear diagonal pattern is observed in most score maps, where the diagonal elements have large values, while most others are close to zero. This observation suggests that the sparse self-attention operation can be accelerated by leveraging this regular attention pattern.

This paper proposes ***Zebra***, a ViT hardware accelerator framework that efficiently utilizes diagonal sparse patterns in self-attention layers, referred to as *Striped Diagonal (SD) pattern*. Zebra is built on three key components: *SD Pruning*, *Striped Diagonal Matrix Multiplication (SDMM)*, and

978-3-9826741-0-0/DATE25/© 2025 EDAA

reconfigurable Zebra accelerator. By leveraging the regular diagonal pattern, *SD pruning* generates the pruning mask by only using three hyper-parameters. This method effectively prunes the SD pattern matrix while minimizing accuracy loss, with tuned parameters both offline and online. Using the pruning mask, *SDMM* computes only the non-zero values in the diagonal elements of the score map and then uses the values to compute the final attention score map. With this efficient computation, SDMM reduces the computational load of the attention mechanism and maximizes data reuse. *Zebra accelerator architecture* is designed to support the SDMM effectively with its reconfigurable architecture that consists of PE lines, diagonal controller, and small SRAM buffers.

Our comprehensive evaluation shows that Zebra achieves an average speedup of 56.6× over a CPU, 12× over an Edge GPU, 1.7× over the state-of-the-art ViT accelerator (ViTCOD [13]), and 1.6× over a GPU, with only a 8.7% hardware overhead from additional components. This significant performance gain is primarily due to Zebra's high PE utilization and its effective SD pruning scheme. Zebra achieves up to 99% PE utilization, while the state-of-the-art ViT accelerator [13] reaches an average of only 53%. In addition to this improvement in PE utilization, Zebra's SD pruning maintains inference accuracy comparable to ideal magnitude-based pruning.

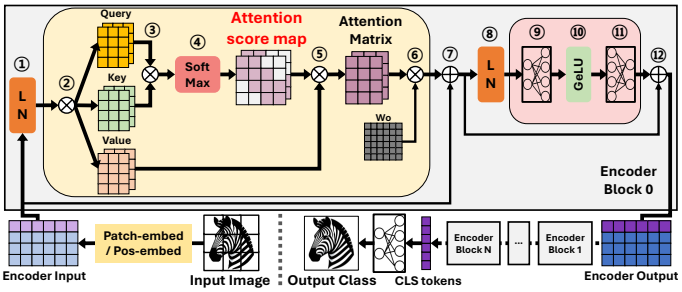


Fig. 2: Operation flow of vision transformer.

II. Background and Motivation

A. Vision Transformer and Attention Mechanism

Figure 2 illustrates the key operations of a typical Vision Transformer (ViT) model. The input image is first divided into multiple patches which are then passed through an embedding operation. The computation proceeds through several encoder blocks, each of which represents a layer of the ViT model. Each block consists of two main operations: attention (②-⑥) and Multi-Layer Perceptron (MLP) (⑨-⑪), along with others (e.g., layer normalization (①,⑧), residual connection (⑦,⑫)).

The attention mechanism, a fundamental operation in ViTs, involves four key steps. First, the query (Q), key (K), and value (V) matrices are generated from the input of the encoder and linear layer weights (②). Then, the attention score map, which captures the relevance between tokens, is computed by taking the dot product of the Q and K matrices (③). Next, the score map is normalized using a row-wise SoftMax function to convert the scores into probabilities (④). Finally, the attention output (A) matrix is produced by multiplying the normalized score (S) matrix with the V matrix (⑤).

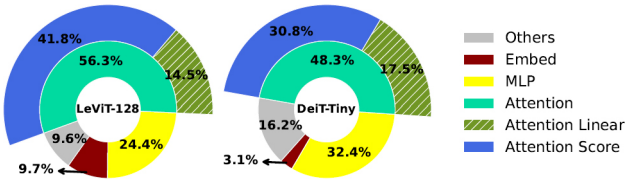


Fig. 3: Latency breakdown of ViT models on edge GPU [18].

It is well known that the computational complexity of self-attention (SA) in transformers grows quadratically with respect to the sequence length (or the total number of patches from input image) [19]. Additionally, unlike basic matrix multiplication, SA is difficult to accelerate using conventional parallel processors (e.g., GPU [20], Systolic array [21]) due to hardware-unfriendly operations like matrix transpose and SoftMax, as well as the substantial memory accesses [12]. Figure 3 shows the breakdown of inference latency for representative ViT models [16], [22], measured on an Edge GPU [18]. In the figure, *Attention Score* refers to the operations producing the attention score map, including QK computation, SoftMax, and SV computation. Attention Linear includes both the QKV projection and output projection. The experimental results indicate that the attention score operations (shown in blue) contribute the most to overall latency. Specifically, attention score operations account for 74.2% and 63.8% of the latency in all attention operations (including attention score operations and attention linear operations) for LeViT-128 [22] and DeiT-Tiny [16], respectively. In terms of total end-to-end inference latency, the attention score operations contribute 41.8% for LeVit-128 and 30.8% for Deit-Tiny.

Figure 4 shows the relative computational efficiency of key operations in ViT inference for various ViT models. The computational efficiency is calculated as the ratio of FLOPs (Floating Point Operations) to the real execution time on an Edge GPU [18]. A value greater than 1 indicates higher computational efficiency compared to the overall operations, whereas a value less than 1 reflects relatively lower efficiency. For all models, MLP and attention linear operations demonstrate values greater than 1, indicating superior computational efficiency. In contrast, attention score computations exhibit an efficiency of less than 0.5, indicating that they are highly inefficient in execution on the parallel hardware relative to the average. This analysis result confirms that the SA mechanism, particularly the attention score computation, is a major bottleneck in accelerating ViT inference on parallel processors. Therefore, this paper proposes a novel hardware accelerator specifically designed to target attention score computations, which not only exhibit the highest latency but also have very poor computational efficiency.

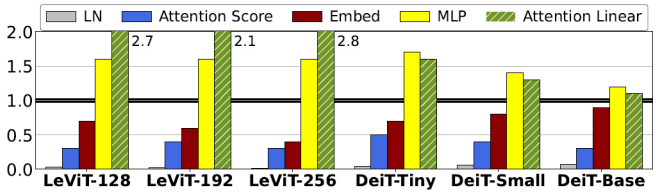


Fig. 4: Relative computational efficiency for ViT operations.

TABLE I: Prior transformer sparse accelerator

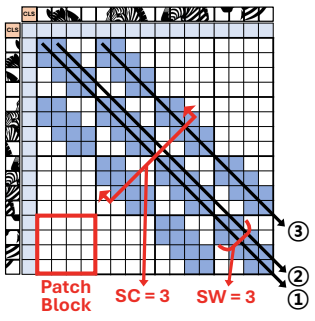
	Sanger [24]	SpAtten [23]	HeatViT [12]	ViTCOD [13]	Zebra (This Work)
Application field	NLP	NLP	ViT	ViT	ViT
Preprocessing	Dynamic	Dynamic	Dynamic	Static	Both
Metadata size	Moderate	Low	Moderate	High	Very Low
Technique Coverage	100 %	75 %	75 %	35 %	85%
Sparsity	Moderate	Low	Moderate	Very High	High
PE utilization	Low	High	High	Low	Very High

B. Prior Transformer Accelerators

In general, most input tokens are irrelevant to the current token being processed, which leads to the attention mechanism inherently exhibiting a high sparsity [17]. To improve the computational efficiency of SA modules, several prior works have proposed the use of sparse attention, as summarized in Table I. Spatten [23] dynamically removes unnecessary input tokens and attention heads, resulting in only moderate sparsity. Sanger [24] adopts low precision to predict dynamic sparse attention mask through a reconfigurable architecture. However, NLP transformer accelerators struggle to achieve high levels of sparsity due to the overhead of dynamic sparsity prediction. HeatViT [12], a ViT accelerator, introduces an additional layer that dynamically calculates the importance of input tokens to eliminate irrelevant ones. All of these accelerators rely on dynamic input-dependent sparse mask prediction, which is inefficient for ViT models with static inputs. ViTCOD [13] proposed utilizing fixed patterns in ViT to polarize workloads into sparser and denser regions, enabling reduced computation. However, ViTCoD requires substantial metadata and model inference must be performed in advance to apply fine-grained fixed patterns. Moreover, the sparser regions are much larger than the denser regions in most layers, which leads to low PE utilization due to the irregular sparse pattern. Therefore, our work, Zebra, proposes a pruning algorithm that leverages a fixed diagonal sparse pattern, which is a more dominant sparsity pattern in ViT, to achieve high sparsity levels while minimizing metadata, along with a reconfigurable architecture to enhance hardware utilization.

C. Striped Diagonal Pattern

Prior works have shown that transformers often exhibit a diagonal attention pattern [9], [14], [17]. Similarly, ViTs exhibit *Striped Diagonal (SD) pattern* on their attention score map, where high score values constitute multiple diagonal stripes, and the rest of them are near zero. This pattern happens mainly because of the strong correlation between the adjacent patches in four directions (top, bottom, left, right). Figure 5 shows an example attention score map exhibiting the SD pattern. The centermost diagonal line (①) indicates the attention score with itself for each patch. The line like ② on the left/right side of the central diagonal stands for the attention scores of the patches on the left and right, respectively. The distant diagonal lines like ③ represent the attention scores with the patches located above and below each patch. The distance between the central diagonal line (①) and its adjacent diagonal lines (③) is determined by the number of patches in the input image.



(a) Hyper-parameters

(b) Input patches

Fig. 5: Striped diagonal pattern and its hyper-parameters.

Since ViTs have a fixed number of input patches, we regularize the SD pattern of the attention score map using three parameters. The *Patch Block (PB)* stands for a cluster of score maps that exhibits a single diagonal stripe pattern repeating over the entire map. The size of PB is the same as the number of patches in the row and column of the input image. The *Stripe Width (SW)* represents the width of the diagonal stripe, and *Stripe Count (SC)* determines the number of diagonal stripes.

III. Zebra

Zebra accelerator framework employs three key components: *SD Pruning method*, *Striped Diagonal Matrix Multiplication (SDMM)*, and *reconfigurable Zebra architecture*. In Section III-A, we describe the SD pruning method that leverages the striped diagonal pattern in ViTs to effectively reduce computation while minimizing accuracy drop. Next, in Section III-B, we introduce SDMM optimized for efficiently processing sparse striped diagonal matrices on hardware. Finally, in section III-C, we introduce the Zebra architecture that incorporates a reconfigurable adder tree to support the matrix multiplication method.

A. Striped Diagonal Pruning (SD pruning)

As described in Section II-C, the attention score map shows a diagonal pattern, and it is plausible to reduce the computational intensity of the ViT by dynamically skipping computations related to small attention scores. Therefore, we propose *Striped Diagonal pruning (SD pruning)* to convert near-zero attention scores into zeros. The Zebra framework determines the eligible sparsity level of the attention score map offline with negligible accuracy degradation, as described in Section IV-B. It then selects the hyper-parameters PB, SW, and SC based on the sparsity level, as these parameters directly define the attention map's sparsity. The framework calculates the column index of each non-zero (NZ) attention score using these hyper-parameters. Since NZ attention scores are critical to the inference result, the hyper-parameters are defined to maximize the sum of NZ scores per row, minimizing accuracy degradation.

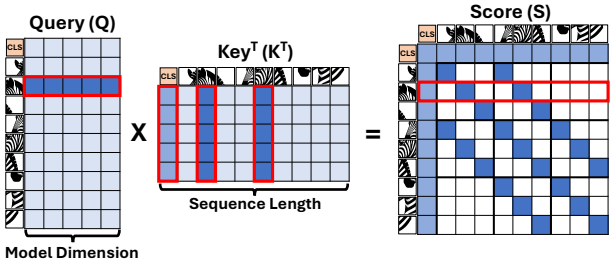
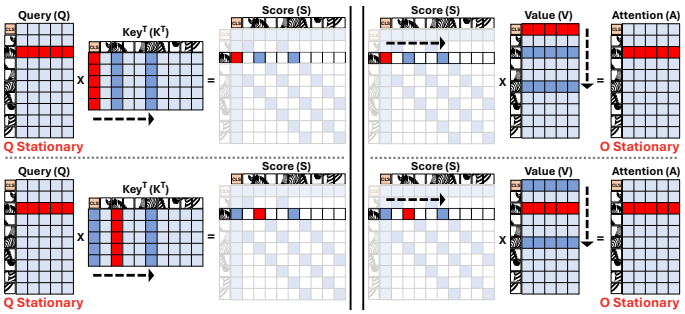


Fig. 6: Matrix multiplication with column skipping



(a) Inner product with Q stationary for QK-MM (b) Row-wise inner product with output stationary for SV-MM

Fig. 7: Striped diagonal matrix multiplication (SDMM).

B. SDMM (Striped Diagonal Matrix Multiplication)

Figure 6 illustrates a *Column Skipping* scheme, which plays a crucial role in SDMM by reducing the computational complexity of *Attention Score* operation. This scheme skips specific columns of the Key matrix (K) during multiplication. To compute a row in the S matrix, the conventional matrix multiplication method requires the corresponding row vector of the Query matrix (Q) and the entire K matrix. In contrast, the column skipping uses a row vector of the Q matrix and only a subset of column vectors of the K matrix (highlighted in red) to produce a row of the S matrix, as it calculates only the diagonal elements of the S matrix.

The SDMM involves two computational stages, as shown in Figure 7. The first stage is *QK matrix multiplication (QK-MM)*, where dense Q and K matrices are multiplied to produce a sparse diagonal S matrix. The second stage is *SV matrix multiplication (SV-MM)*, where the sparse diagonal S matrix is multiplied by the dense V matrix.

For QK-MM, the SDMM efficiently reduces computation with the column skipping described above. To maximize Q data reuse, it employs an *Inner product with Q stationary* approach. Due to the row-wise SoftMax in the attention mechanism, each input token (each row of the Q matrix) must have at least one K token that generates NZ score elements. This means that a single Q row is calculated with multiple K columns to generate one S row, and the Q matrix's data reuse can be maximized through Q stationary. As illustrated in Figure 7a, a single Q row is reused for multiple necessary columns of K to compute a row of the sparse diagonal S matrix.

For SV-MM, the SDMM employs a *Row-wise inner product with Output stationary* approach, as illustrated in Figure 7b. All the NZ elements in a single row of the S matrix are multiplied by the corresponding rows of the V matrix, where the row index of the V matrix is determined by the column index of the NZ value in the S matrix. The product results are then accumulated in the corresponding row of the attention (A) matrix. To minimize memory access for the partial sums, the SDMM utilizes an output stationary approach. The NZ values of the S matrix and the corresponding row vectors of the V matrix are processed sequentially together, and the partial sums of the rows are accumulated in the Adder Tree's internal buffer (the detailed architecture of the Adder Tree is thoroughly described in Section III-C3).

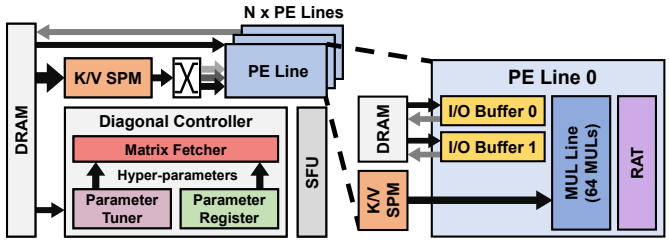


Fig. 8: Overall architecture of Zebra accelerator

C. Zebra Accelerator

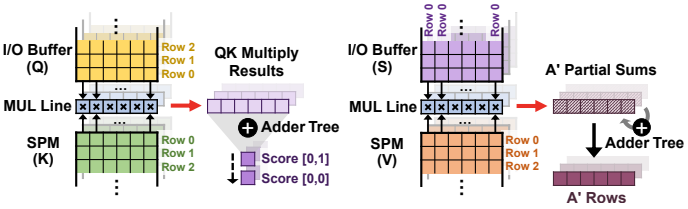
*1) **Overview**:* Figure 8 shows the overview of Zebra accelerator architecture, incorporating Processing Element (PE) lines, a K/V Scratch Pad Memory (SPM), and a diagonal controller.

Each *PE line* consists of two I/O buffers, a Multiply (MUL) line, and a Reconfigurable Adder Tree (RAT). The I/O buffers are dedicated to the Q, A, and S matrices, functioning as bidirectional FIFO buffers. They store the Q matrix loaded from DRAM and the S or A matrix computed within the PE line. Since the elements of these matrices are used only once during the computation and do not need to be retained for future operations, small-sized I/O buffers are more than sufficient for temporary storage. The MUL line consists of 64 multipliers that perform simple dot product operations using data from one of the I/O buffers and the K/V SPM, and then sends the computation results to the adder tree (Section III-C2). The adder tree is designed with a reconfigurable architecture to support two stages of SDMM (i.e., QK-MM and SV-MM), minimizing the number of additional adders (Section III-C3).

The *K/V SPM* stores the K or V matrices required for the calculation of the score map. As these matrices are reused multiple times, storing them in the SPM reduces the data movement overhead of DRAM. Since only the NZ values of the S matrix are stored in DRAM during SDMM, no additional metadata is required to track their location in the S matrix.

The *diagonal controller* computes the indices using hyper-parameters SW, SC, PB to fetch the necessary matrix data from memory during the SDMM. The hyper-parameters can be either precomputed offline or tuned online in parameter tuner (Section III-C4). Additionally, Zebra employs a Special Function Unit (SFU) for specific operations such as SoftMax.

*2) **MUL Lines and Data Mapping**:* Figure 9 illustrates the SDMM computations (QK-MM and SV-MM) in Zebra's PE. As described in section III-B, SDMM employs inner product with Q stationary for QK-MM. To support Q stationary, the MUL line reuses a row of Q matrix multiple times for computations with all the corresponding rows of K matrix without transposing K. Each MUL line receives the pre-stored Q row vector from the I/O FIFO buffer. It fetches one row of K matrix per cycle

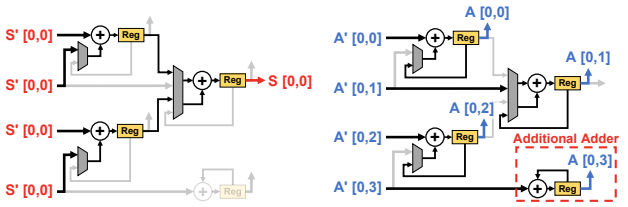


(a) PE execution for QK-MM (b) PE execution for SV-MM

Fig. 9: PE execution for SDMM.

from the K/V SPM to perform inner product with the Q row. Since Q, K, and V row vectors are 64 elements long, Zebra uses 64 multipliers per MUL line. The result from one MUL line is passed to the adder tree, summed to produce one element of the score matrix. As all Q and K computations are independent, all MUL lines of PE lines can be processed in parallel, allowing multiple Q rows to be processed simultaneously.

For SV-MM, SDMM employs a row-wise inner product, where each element of S matrix is multiplied by the corresponding row vector of V matrix (i.e., scalar multiplication). To support this dataflow, a Score element is duplicated across an I/O buffer entry. This ensures that the same S element is delivered to all 64 multipliers of the MUL line, as shown in Figure 9b. With this mapping, one row of S matrix is transposed and processed per MUL line at a time. At every cycle, the MUL line reads a row vector of the V matrix from the SPM and multiplies it by an element S. This multiplication results in a partial sum of a row in A matrix, which is accumulated by the adder tree to produce a complete row of A matrix. As in QK-MM, each MUL line operates independently, enabling the computation of multiple rows of S matrix in parallel.



(a) Configuration for QK-MM (b) Configuration for SV-MM

Fig. 10: Reconfigurable adder tree.

*3) **Reconfigurable Adder Tree**:* In Zebra architecture, the adder tree supports the two SDMM stages (QK-MM and SV-MM) through a reconfigurable architecture for optimized accumulation dataflows. In QK-MM, 64 multiplication results are summed into a single Score element, which can be efficiently computed using a conventional adder tree structure. In SV-MM, the partial sums of a row are summed element-wise to produce one complete row of A matrix. We can support this accumulation process of the partial sums by adding only a single adder and additional paths to the conventional adder tree.

Figure 10 illustrates the Reconfigurable Adder Tree (RAT) architecture (4-input RAT are shown for ease of explanation). The RAT includes accumulator registers to hold the output of the adders and multiplexers to change the input source for the adders. As shown in Figure 10a, the RAT works as the conventional adder tree for QK-MM. In contrast, for SV-MM, the RAT is configured to select the accumulator register as one of the adders' inputs (Figure 10b). With this configuration, the RAT accumulates the newly calculated partial sum into the accumulator registers. After accumulating all partial sums, the row of the A matrix is stored in the I/O buffer.

*4) **Diagonal Controller**:* Thanks to SD pruning and pipelined SDMM, the diagonal controller can easily calculate the indices of score map in real time for each row processed by each MUL line, using only basic addition and subtraction operations. As shown in Figure 8, a matrix fetcher in the

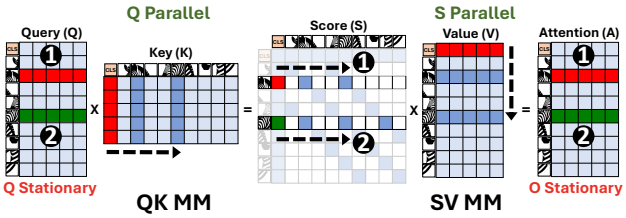


Fig. 11: Parallelism of SDMM for data reuse of input matrix.

diagonal controller receives the hyper-parameters (PB, SC, and SW) from a parameter tuner or a parameter register. It then calculates indices of non-zero attention scores for each row by incrementally adjusting left and right from the diagonal element's index based on the parameter values. The parameter register stores predefined parameter values. Using SD pattern, the parameter tuner updates the hyper-parameters SC and SW for the next layer based on the cumulative sum of just one score row. By adjusting SC and SW, it controls the thickness of the diagonal stripe and computes the cumulative sum of the corresponding elements in a score row. Based on this sum, it tunes SW and SC by increasing or decreasing them as needed.

Zebra maximizes data reuse for the Q, S, and A matrices through Q stationary and Output stationary in SDMM. To avoid multiple access to the same row of K and V matrices, which can result in bank conflicts in the K/V SPM, we propose a parallelism method as shown in figure 11. Since the SD matrix repeats the same pattern across patch blocks and forms a thick diagonal shape, multiple rows of the Q matrix (like ❶, ❷) can be multiplied by the same K column and processed simultaneously. In the same way, multiple rows of the S matrix can be multiplied by the same row of the V matrix. The diagonal controller distributes data from the SPM to MUL lines sharing the same vector via control signal for an crossbar. The crossbar is connected to each bank of the SPM and individual MUL lines, enabling fully parallel processing in SDMM.

IV. Evaluation

A. Experimental Methodology

Benchmarks. We evaluate Zebra's performance on the ImageNet [25] dataset for image classification, using DeiT-Tiny/Base/Small and LeViT-128/192/256 models.

Baselines. We compare the Zebra with 4 baselines including 3 real hardware platforms: CPU (AMD EPYC 75F3), Edge GPU (Jetson AGX Xavier), GPU (NVIDIA RTX 3090), and the state-of-the-art sparse ViT accelerator (ViTCOD [13]). We measure CPU performance using PyTorch Profiler [26], GPU performance with NVIDIA Nsight Systems [27], and ViTCoD performance using a cycle-based simulator from the paper.

Performance Simulation. We use an in-house cycle-accurate simulator to evaluate the performance of the Zebra accelerator. As described in Section II, we focus on measuring the execution time of the attention score computation, which

TABLE II: Zebra hardware configuration.

DRAM		25GB/s LPDDR5
SRAM	K/V SPM	16KB, 8 bank
	I/O Buffer	1KB & 64B, 8 bank
PE	MUL line	512 multipliers (8 lines with each 64)
	Adder tree	512 adders (511+1)

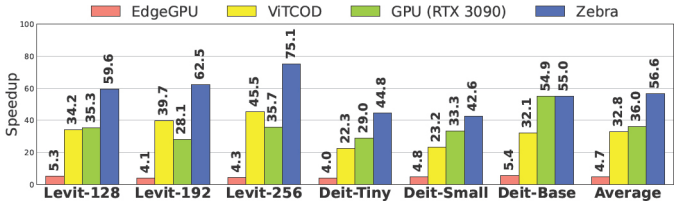


Fig. 12: Performance improvement over CPU.

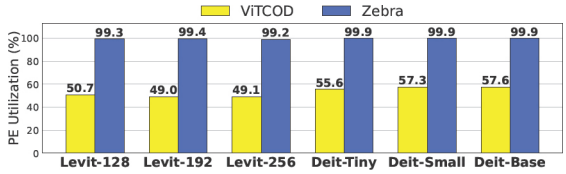


Fig. 13: PE utilization of Zebra and ViTCoD [13].

is the primary performance bottleneck. We pruned the ViT models to a 60% sparsity level using magnitude-based pruning for ViTCoD, and applied SD pruning for Zebra in the same accuracy for a fair comparison. The sparsity level does not severely reduce the model accuracy, as shown in Figure 14.

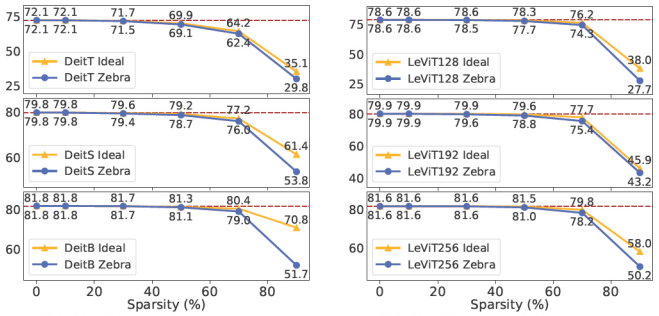
Implementation. We implement an RTL design of Zebra in Verilog HDL and synthesize it using Synopsys Design Compiler with UMC 28nm library at 500MHz. Table II presents the detailed configuration of Zebra accelerator used in the evaluation. The area of SRAM buffers is estimated with CACTI 7.0 [28]. The SoftMax unit is implemented using Base-2 [29].

B. Experimental Results

Performance. Figure 12 shows overall performance of Zebra and four baselines on core attention computations. On average, Zebra achieves 56.6×, 12×, 1.7×, and 1.6× performance improvements over the CPU, Edge GPU, ViTCOD, and GPU platforms. Zebra shows slightly different inference performance across models depending on each model's SD pattern ratio, but achieves superior performance in all ViT models. For a fair comparison, we fixed CPU threads to 4 for all models. For the GPU, we consider only on-device data movement and kernel execution time, excluding host memory copy time.

PE Utilization. This significant performance improvement of Zebra is mainly due to its high PE utilization. Zebra performs SDMM to compute SD matrix on hardware efficiently, and its architecture is optimally designed to maximize PE utilization. Figure 13 shows PE utilization for both ViTCOD and Zebra during attention score computation. In Zebra, all MUL lines are independently mapped to rows and columns, ensuring that no multipliers are left idle. Additionally, the adder tree is fully pipelined with the MUL lines, thanks to the registers in each adder. Consequently, Zebra achieves 99% of PE utilization regardless of the size and sparsity of the attention score map.

Inference Accuracy. Figure 14 shows the inference accuracy at various sparsity levels for several ViT models. We compare Zebra's SD pruning with the ideal magnitude-based pruning used in ViTCOD. Up to 50% sparsity, there is virtually no difference between the ideal pruning and SD pruning. At very high sparsity levels (above 80%), SD pruning causes a slightly larger accuracy drop, though even ideal pruning suffers a substantial drop compared to the vanilla model. At reasonable sparsity levels (around 60%), there is no significant difference



(a) DeiT model accuracy (b) LeViT model accuracy

Fig. 14: Inference accuracy for various pruned ViT models.

between ideal and SD pruning, demonstrating that the SD pruning can achieve high sparsity with minimal accuracy loss.

Hardware Area. We evaluated the hardware area focusing on the key components of Zebra architecture. As shown in Table III, Zebra occupies a total area of 0.73 mm^2. The additional hardware elements, including the additional adders in the Adder Tree and the diagonal controller, contribute an overhead of only 8.7% of the total area. Despite using sparse pattern, SDMM and the reconfigurable architecture allow only a single access to the Q, S, and A rows, no need to store the entire input matrices in large SPM. As a result, Zebra only has about 17KB SRAM for K/V SPM and FIFO buffer, which helps hardware area's compact. In contrast, ViTCoD with total area $3mm^2$ needs to access the same data multiple times, requiring a much larger 320KB SPM, about 19 times larger [13].

TABLE III: Breakdown of hardware area

PE		**Diagonal Controller**	**SRAM**		**SFU**
MUL	Adder tree		K/V SPM	I/O buffer	SoftMax
7.73%	5.06% (0.07%)*	8.67%	75.48%	2.39%	0.67%
12.79%			77.87%		
0.73 mm^2					

* Additional Adder

V. Conclusion

ViTs have a Striped Diagonal (SD) attention pattern since information near each patch token is the most critical. This paper introduces a Zebra accelerator framework optimized for sparse attention computation with diagonal patterns. Zebra incorporates three key components: an efficient diagonal pruning technique, a Striped Diagonal Matrix Multiplication (SDMM), and a reconfigurable architecture to effectively support the SDMM. Our experimental results shows that Zebra achieves 99% PE utilization and speedup of 57× compared to a CPU.

VI. Acknowledgment

This work was partly supported by Institute of Information & communications Technology Planning & Evaluation(IITP) grant funded by the Korea government(MSIT) (No.00228970, 50%), the Ministry of Science and ICT (MSIT) under the Information Technology Research Center (ITRC) support program (No.II212052, 20%) supervised by the Institute for Information & Communications Technology Planning & Evaluation (IITP) and Korea Institute for Advancement of Technology(KIAT) grant funded by the Korea Government(MOTIE) (No.P0023704, 30%). The EDA tool was supported by the IC Design Education Center (IDEC), Korea. Seokin Hong is the corresponding author.

REFERENCES

[1] A. Vaswani, "Attention is all you need," *Advances in Neural Information Processing Systems*, 2017.

[2] Z. Wu, Z. Liu, J. Lin, Y. Lin, and S. Han, "Lite transformer with long-short range attention," *arXiv preprint arXiv:2004.11886*, 2020.

[3] J. Devlin, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[4] A. Dosovitskiy, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[5] Y. Fang, B. Liao, X. Wang, J. Fang, J. Qi, R. Wu, J. Niu, and W. Liu, "You only look at one sequence: Rethinking transformer in vision through object detection," *Advances in Neural Information Processing Systems*, vol. 34, pp. 26 183–26 197, 2021.

[6] B. Wu, C. Xu, X. Dai, A. Wan, P. Zhang, Z. Yan, M. Tomizuka, J. Gonzalez, K. Keutzer, and P. Vajda, "Visual transformers: Token-based image representation and processing for computer vision," *arXiv preprint arXiv:2006.03677*, 2020.

[7] E. Xie, W. Wang, Z. Yu, A. Anandkumar, J. M. Alvarez, and P. Luo, "Segformer: Simple and efficient design for semantic segmentation with transformers," *Advances in neural information processing systems*, vol. 34, pp. 12 077–12 090, 2021.

[8] B. Heo, S. Yun, D. Han, S. Chun, J. Choe, and S. J. Oh, "Rethinking spatial dimensions of vision transformers," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 11 936–11 945.

[9] H. Li, Z. Li, Z. Bai, and T. Mitra, "Asadi: Accelerating sparse attention using diagonal-based in-situ computing," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2024, pp. 774–787.

[10] K. Choromanski, V. Likhosherstov, D. Dohan, X. Song, A. Gane, T. Sarlos, P. Hawkins, J. Davis, A. Mohiuddin, L. Kaiser *et al.*, "Rethinking attention with performers," *arXiv preprint arXiv:2009.14794*, 2020.

[11] J. Dass, S. Wu, H. Shi, C. Li, Z. Ye, Z. Wang, and Y. Lin, "Vitality: Unifying low-rank and sparse approximation for vision transformer acceleration with a linear taylor attention," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2023, pp. 415–428.

[12] P. Dong, M. Sun, A. Lu, Y. Xie, K. Liu, Z. Kong, X. Meng, Z. Li, X. Lin, Z. Fang *et al.*, "Heatvit: Hardware-efficient adaptive token pruning for vision transformers," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2023, pp. 442–455.

[13] H. You, Z. Sun, H. Shi, Z. Yu, Y. Zhao, Y. Zhang, C. Li, B. Li, and Y. Lin, "Vitcod: Vision transformer acceleration via dedicated algorithm and accelerator co-design," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2023, pp. 273–286.

[14] D. Yu, T. Xi, J. Li, B. Li, G. Zhang, H. Feng, J. Han, J. Liu, E. Ding, and J. Wang, "Accelerating vision transformers based on heterogeneous attention patterns," *arXiv preprint arXiv:2310.07664*, 2023.

[15] Z. Kong, P. Dong, X. Ma, X. Meng, W. Niu, M. Sun, X. Shen, G. Yuan, B. Ren, H. Tang *et al.*, "Spvit: Enabling faster vision transformers via latency-aware soft token pruning," in *European conference on computer vision*. Springer, 2022, pp. 620–640.

[16] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou, "Training data-efficient image transformers & distillation through attention," in *International conference on machine learning*. PMLR, 2021, pp. 10 347–10 357.

[17] K. Kim, B. Wu, X. Dai, P. Zhang, Z. Yan, P. Vajda, and S. J. Kim, "Rethinking the self-attention in vision transformers," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 3071–3075.

[18] D. Franklin, "Jetson agx xavier and the new era of autonomous machines webinar," 2020.

[19] I. Beltagy, M. E. Peters, and A. Cohan, "Longformer: The long-document transformer," *arXiv preprint arXiv:2004.05150*, 2020.

[20] G. Byeon, S. Lee, S. Kim, Y. Kim, P. J. Nair, and S. Hong, "Sparseft: Sparsity-aware fault tolerance for reliable cnn inference on gpus," in *2023 32nd International Conference on Parallel Architectures and Compilation Techniques (PACT)*. IEEE, 2023, pp. 337–338.

[21] S. Kim, G. Byeon, S. Kim, H. Kim, and S. Hong, "Conveyor: Towards asynchronous dataflow in systolic array to exploit unstructured sparsity," in *2023 IEEE 41st International Conference on Computer Design (ICCD)*. IEEE, 2023, pp. 423–431.

[22] B. Graham, A. El-Nouby, H. Touvron, P. Stock, A. Joulin, H. Jégou, and M. Douze, "Levit: a vision transformer in convnet's clothing for faster inference," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 12 259–12 269.

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

[24] 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," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 977–991.

[25] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[26] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[27] K. Iyer and J. Kiel, "Gpu debugging and profiling with nvidia parallel nsight," *Game Development Tools*, pp. 303–324, 2016.

[28] R. Balasubramonian, A. B. Kahng, N. Muralimanohar, A. Shafiee, and V. Srinivas, "Cacti 7: New tools for interconnect exploration in innovative off-chip memories," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 14, no. 2, pp. 1–25, 2017.

[29] Y. Zhang, Y. Zhang, L. Peng, L. Quan, S. Zheng, Z. Lu, and H. Chen, "Base-2 softmax function: Suitability for training and efficient hardware implementation," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 69, no. 9, pp. 3605–3618, 2022.