SpARC: Token Similarity-Aware Sparse Attention Transformer Accelerator via Row-wise Clustering

Han Cho, Dongjun Kim, Seung-Eon Hwang, and Jongsun Park
School of Electrical Engineering, Korea University, Seoul, Korea
{myeyeget,djkim010592,eonion56,jongsun}@korea.ac.kr

ABSTRACT

Self-attention mechanisms, the key enabler of transformers' remarkable performance, account for a significant portion of the overall transformer computation. Despite its effectiveness, self-attention inherently contains considerable redundancies, making sparse attention an attractive approach. In this paper, we propose SpARC, a sparse attention transformer accelerator that enhances throughput and energy efficiency by reducing the computational complexity of the self-attention mechanism. Our approach exploits inherent row-level redundancies in transformer attention maps to reduce the overall self-attention computation. By employing row-wise clustering, attention scores are calculated only once per cluster to achieve approximate attention without seriously compromising accuracy. To leverage the high parallelism of the proposed clustering approximate attention, we develop a fully pipelined accelerator with a dedicated memory hierarchy. Experimental results demonstrate that SpARC achieves attention map sparsity levels of 85-90% with negligible accuracy loss. SpARC achieves up to 4× core attention speedup and 6× energy efficiency improvement compared to prior sparse attention transformer accelerators.

1 INTRODUCTION

Transformer [1], renowned for its self-attention mechanism, excels in capturing both global and local context within sequences or images. It has proven to be highly effective across a broad spectrum of tasks, ranging from natural language processing (NLP) [2], [3], [4] to computer vision (CV) tasks [5]. The self-attention mechanism, a core computation of the transformer, calculates relationships between each token in an entire input sequence, determining the significance of individual input tokens. This property of relationship calculation drives the continuous increase in input sequence length for enhanced transformer performance (e.g., BERT:512 length [2], GPT-3:2048 length [4]). However, increasing input sequence length presents a fundamental challenge. While longer input sequences may enhance performance, the computational complexity and memory costs of self-attention computation are quadratically increasing with increasing input sequence length. For example, when scaling the sequence length of GPT-2 [3] from 1K to 16K, the required computation of self-attention skyrockets from 38.7 GFLOPs to 9477.3 GFLOPs, positioning the self-attention mechanism as the primary bottleneck in transformer deployment. This inefficiency highlights the importance of complexity reduction in self-attention to achieve scalable transformers.

To tackle the computational complexity and memory usage issues inherent in self-attention, sparse attention has emerged as a promising approach [6], [7]. Sparse attention aims to alleviate the computational burden of dense attention by selectively skipping computations for token pairs with low relevance, effectively eliminating potential redundancies in the whole self-attention process. This selective computation contributes to scalability by reducing the overall computational overhead, allowing for the processing of longer input sequences without incurring excessive resource demands. However, one of the challenges encountered with this selective computation of sparse attention is the introduction of a dynamic and highly unstructured sparsity pattern [8], [9]. This dynamic sparsity pattern poses difficulties in efficient hardware utilization due to load imbalance and the need for dynamic runtime scheduling.

To address the hardware performance degradation associated with prior sparse attention approaches, this paper introduces SpARC, a novel software-hardware co-design framework to accelerate attention computations through row-wise structured sparse attention, offering a solution to make transformer acceleration more practical and scalable. Our main contributions can be summarized as follows:

- We present a computation efficient token clustering approximate attention that analyzes similarities between tokens, grouping them into clusters to achieve row-level sparsity. This approach reduces the computational complexity by only requiring computations for the centroid of each cluster. The proposed attention row-clustering approximate attention provides flexibility in balancing transformer latency and performance.
- We present the SpARC hardware architecture. SpARC efficiently leverages the data parallelism inherent to the proposed clustered approximate attention through a highly pipelined dedicated accelerator. Additionally, a dedicated memory hierarchy, named cluster centroid cache, is implemented to significantly reduce the DRAM access traffic and latency overhead associated with irregular access patterns.
- SpARC's high performance is demonstrated on three distinct transformer models: GPT-2, BERT, ViT, and five different benchmarks: SQuAD v1.1, MNLI, CLOTH, Wikitext-2, ImageNet-1K. Implementation results show that SpARC constantly achieves an average sparsity of 0.85-0.9 with negligible accuracy loss, a core attention speedup of 6×, and an increase in energy efficiency by 4× compared to prior state-of-the-art transformer accelerators.

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

ACM ISBN 979-8-4007-0601-1/24/06.
https://doi.org/10.1145/3649329.3655936

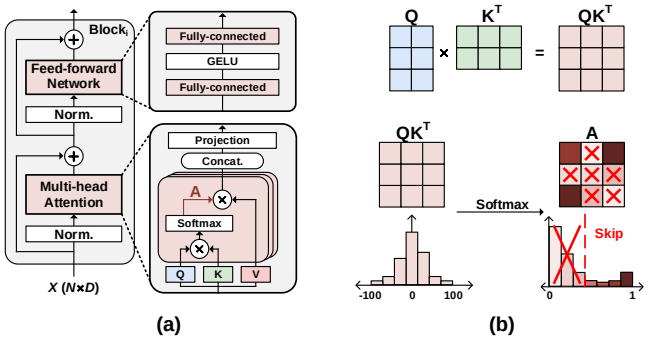

Figure 1: (a) Transformer architecture [1] (b) Prior sparse attention approach

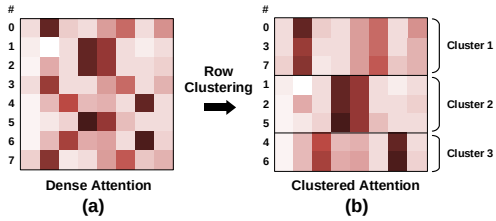

Figure 2: Motivation of the row-clustering approximate attention (a) Sample dense attention (b) Row-clustered attention

2 BACKGROUND

2.1 Transformer

Fig. 1 (a) illustrates the transformer architecture and its main building blocks. Initially, the input sequence of size N is processed by an encoding layer, which transforms the sequence into an input matrix $X \in \mathbb{R}^{N \times D}$. Then, this input matrix X is fed into multiple transformer encoder blocks. Each block comprises two primary components: multi-head self-attention and a feed-forward network (FFN). The multi-head self-attention, or simply self-attention, receives the input X and performs three distinct linear transformations to generate matrices query (Q), key (K), and value (V), each of dimension $\mathbb{R}^{N \times D}$ as follows:

$$Q, K, V = X(W_Q, W_K, W_V). \tag{1}$$

Following the generation of Q, K, and V as shown in Eq. 1, these matrices are split along hidden dimension D to $\frac{D}{h}$, resulting in multiple heads. For each head, the attention map A is expressed as

$$A = Softmax(\frac{QK^T}{\sqrt{d_k}}), \quad A \in \mathbb{R}^{N \times N}. \tag{2}$$

Then, the attention matrix A of each head is multiplied with its corresponding V to obtain $Z = AV$. This Z matrix is then concatenated, linearly projected, and passed through a residual connection before entering FFN. Within FFN, the input Z matrix undergoes a series of transformations, including a fully-connected layer, a GELU activation function, another fully-connected layer, and a residual connection. This process is repeated for each transformer block within the overall transformer architecture.

2.2 Sparse Attention

The self-attention mechanism in transformers poses a significant bottleneck in their deployment due to its quadratic complexity increase with respect to the input sequence length. Specifically, when N is the input sequence length and D is the hidden dimension, the matrix multiplication, scaling, and Softmax of QK^T in the self-attention module incurs a computational complexity of $O(N^2D)$. In contrast, the remaining matrix multiplications of Eq. 1, $Z = AV$, and FFN involve $O(ND^2)$ computations. Considering the trend of increasing input sequence lengths in transformers, this $O(N^2D)$ complexity necessitates the development of efficient attention computation techniques.

To alleviate the computational burden associated with self attention, sparse attention offers a promising solution. While the attention mechanism aims to capture relationships between tokens in the input sequence, not all tokens exhibit strong relationships. For instance, tokens like articles 'a' or 'the' may not significantly impact sequence processing and can yield near-zero outputs from the Softmax function in Eq. 2. Shown in Fig. 1 (b), prior works [6], [7] exploit this observation to reduce computational complexity by omitting calculations for near-zero values of the element-wise sparse attention map. However, this element-wise sparsity poses challenges for efficient hardware utilization due to the dynamic nature of the sparsity pattern during runtime. A hardware-friendly structured sparse attention mechanism is highly required to leverage the computational benefits of sparse attention fully.

3 PROPOSED ROW-CLUSTERING APPROXIMATE ATTENTION

In this section, we introduce the row-clustering approximate attention method aimed at reducing the computational complexity of self-attention based on the row-wise similarity of the attention map.

3.1 Motivation

Attention Map Row-wise Redundancy: The self-attention mechanism, a key component of transformer architectures, is computationally expensive due to its inherent redundancies [7], [10]. Addressing these redundancies is a promising approach to improving the efficiency of transformers. As discussed in Section 2.2, unstructured element-wise sparsity eliminates element-wise redundancy and achieves high sparsity levels. However, it incurs significant hardware overhead due to the dynamic nature of sparsity, requiring complex interconnects, non-zero matching, and load balancing mechanisms [11]. The hardware overheads associated with element-wise sparsity motivate the exploration of a coarser-grained approach to sparsity, namely row-wise clustering.

Fig. 2 (a) shows an example of a dense attention map that appears to be random, not exhibiting any specific patterns. However, upon analyzing individual rows, we can identify patterns of similarity between them. For example, in Fig. 2 (a), it is evident that the dense attention map rows of {0,3,7}, {1,2,5}, and {4,6} show similar patterns. These similar row patterns motivate row-clustering, where the rows with similar patterns are grouped together, as illustrated in Fig. 2 (b). Our core idea is to approximate the entire attention map by calculating attention scores only once per *representative cluster row (i.e., centroid)* instead of row by row. This approach

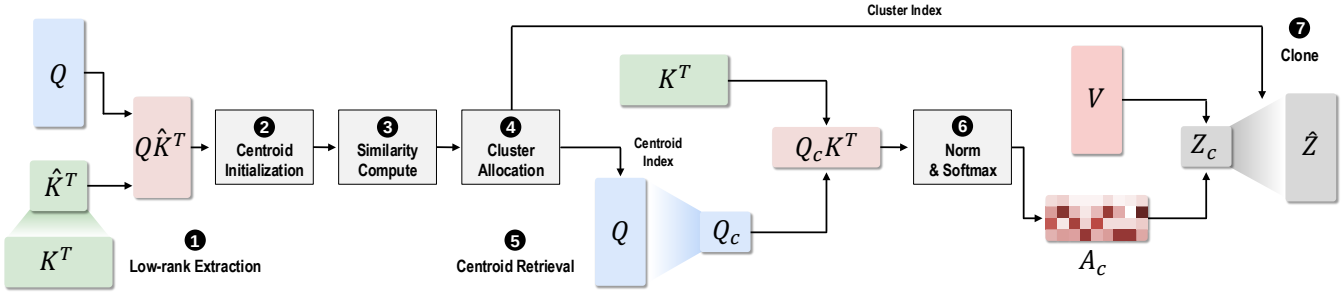

Figure 3: Proposed row-clustering approximate attention

stems from the definition of a centroid: the data point within a cluster that exhibits the highest degree of similarity to all other data. Our approximate self-attention mechanism significantly reduces computational complexity in attention computation with minor accuracy degradation. Furthermore, this row-wise clustering-based approach offers flexibility in balancing computational reduction with performance by adjusting the number of clusters.

3.2 Proposed Clustering Approximate Attention

Fig. 3 illustrates the overall process of the proposed row-clustering approximate attention. In the example shown in Fig. 2, row clustering is performed after dense attention computation finishes. However, this does not reduce the quadratic computation of dense attention. To reduce the $O(N^2D)$ dense attention computations, row-clustering should be performed prior to the dense attention map A. So, we first introduce approximating the dense attention matrix A through a low-rank extraction of QK^T for the following row-clustering.

Low-rank Extraction: ❶ in Fig. 3 shows the low-rank extraction process. Given $Q, K \in \mathbb{R}^{N \times \frac{D}{h}}$ (far left of Fig. 3), a low-rank extraction of K^T is performed to obtain a smaller matrix $\hat{K^T}$. Then, the low-rank $\hat{K^T}$ is multiplied with Q to obtain a low-rank approximation $Q\hat{K}^T \in \mathbb{R}^{N \times N_{low}}$. The specific columns to be extracted are determined offline through linear discriminant analysis (LDA).

Centroid Initialization: Following the computation of $Q\hat{K}^T$, to facilitate the following row similarity computations, C initial centroids are first computed [12]. As shown in ❷ of Fig. 3, C centroids are initialized uniformly in space for similarity computation of $Q\hat{K}^T$ rows. The initial centroid matrix $Centroid \in \mathbb{R}^{C \times N_{low}}$ is defined as follows:

$$Centroid_c = \frac{\sum_{j \in A_c} Q\hat{K}^T_{ij}}{|A_c|}, \tag{3}$$

where A_c refers to the set of rows required to compute centroid c. This initialized centroid matrix will be updated later on.

Similarity Compute: Once the centroids are initialized, the pairwise L1 distance (❸ in Fig. 3) is computed between each row of $Q\hat{K}^T$ and centroids in $Centroid$. Previously calculated initial centroids allow for a low-cost similarity computation by only comparing each row with limited centroids. L1 distance is chosen as a similarity metric due to its low computation overhead, avoiding computationally expensive multiplications. The resulting similarity matrix $Sim \in \mathbb{R}^{N \times C}$ is then defined as follows:

$$Sim_{ij} = ||Q\hat{K}^T_i - Centroid_j||_1. \tag{4}$$

Cluster Allocation: Following the computation of the pairwise distance matrix Sim, this matrix is used to assign each row of the attention map to its corresponding cluster (❹ in Fig. 3). The cluster number for row i is defined as follows:

$$Cluster_i = \underset{i}{\operatorname{argmin}}\, Sim_i. \tag{5}$$

Once the allocation of each row to its corresponding cluster is complete, the similarity matrix is traversed column-wise to identify the row that best represents the centroid for each cluster. Subsequently, the obtained row indices, the rows with the highest similarity with other rows within a cluster, are retrieved from Q to obtain $Q_c \in \mathbb{R}^{C \times \frac{D}{h}}$ (❺ of Fig. 3). This Q_c is used to perform lightweight attention computations, including Q_cK^T, normalization, Softmax of ❻, and $Z_c = A_cV$, significantly reducing the overall computational complexity.

Clone: Following the completion of all the computations up to $Z_c = A_cV$, the resulting matrix $Z_c \in \mathbb{R}^{C \times \frac{D}{h}}$ is concatenated and expanded back to its original dimension (❼ in Fig. 3) of $\mathbb{R}^{N \times D}$ matrix size to perform the linear projection and the subsequent feed-forward network. The generation of approximated $\hat{Z}$ expansion involves replacing all rows belonging to a particular cluster with the corresponding cluster centroid.

3.3 Number of Clusters

Balancing accuracy and computation reduction requires careful consideration of the number of clusters used for the approximate attention computation. While a larger number of clusters and varying cluster numbers for each transformer block reduce the approximation errors, complexity reduction in attention computations decreases. For an effective cluster number selection, we predetermine the cluster numbers for each transformer block by considering a trade-off between accuracy and computational complexity. For gauging the error introduced by clustering, we employ the Kullback-Leibler (KL) divergence as a guiding metric to compare the attention map before and after clustering. We consider the error between the original attention matrix A and the clustered attention matrix after cloning $\hat{A}$ as follows:

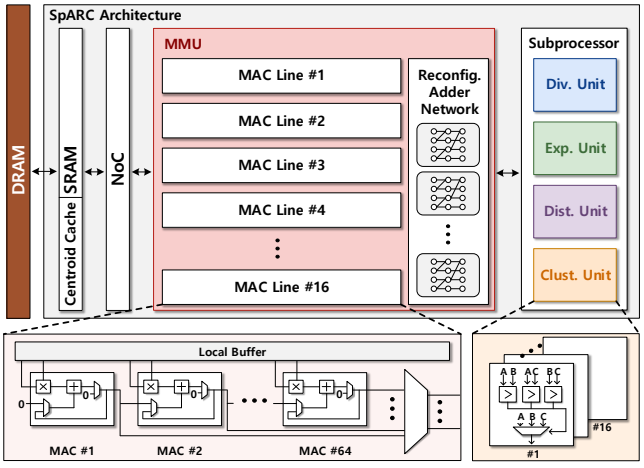

Figure 4: SpARC hardware architecture

$$error = \sum_{i=0}^{N} KL(A_i, \hat{A}_i). \tag{6}$$

By employing this metric, we can evaluate clustering performance in relation to the number of clusters. By establishing a global error threshold t for each transformer block, we can identify the minimum number of clusters that meet the specified error threshold such that $error \leq t$. This global thresholding approach allows the number of clusters to vary across transformer blocks. For instance, a threshold of 0.005 for GPT-2 results in a cluster number of 218 for block 1 and 43 for block 7, accommodating the varying sensitivity of transformer blocks. Consequently, our row-clustering approximate attention balances performance and the number of clusters (attention row sparsity).

4 SpARC HARDWARE ARCHITECTURE

4.1 Motivation

As mentioned, increasing input sequence length becomes the primary bottleneck in transformer deployment due to quadratically increasing computational complexities. The proposed row-clustering approximate attention effectively addresses the attention computation bottleneck by reducing the sequence length (N) to a smaller parameter representing the number of clusters (C) in $O(N^2D)$. While this method significantly reduces the computational complexity of the self-attention mechanism, existing general computing platforms, including CPU and GPU, are not well-suited for accelerating the proposed clustering approximate attention. This limitation stems from two primary factors. First, general computing platforms are not optimized for the specific operations involved in the row-clustering approximate attention, such as absolute distance and minimum distance locator computations. These operations are fundamental to the clustering process, and their inefficient execution on general platforms hinders overall performance. Second, general computing platforms face challenges in exploiting data parallelism effectively for the proposed row-clustering approximate attention algorithm. The inherent data dependencies within the clustering process limit the ability to parallelize computations across multiple processing units, thereby restricting the performance gains achievable on these platforms. To overcome these limitations of general computing platforms and harness the potential of the proposed row-clustering approximate attention, we propose a dedicated accelerator that will be presented in the following section.

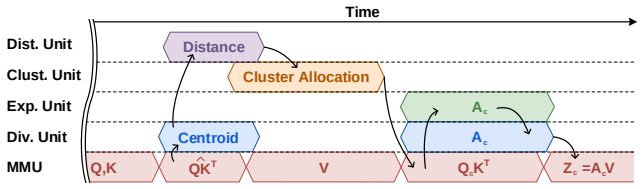

Figure 5: SpARC computation pipeline

4.2 Architecture Overview

In this section, we present SpARC, a dedicated accelerator specifically designed for row-clustering based approximate attention. To handle the unique computational requirements of the approximate attention, SpARC is equipped with a central compute core comprising a matrix multiply unit (MMU) and multiple sub-processors. Fig. 4 shows the overall SpARC hardware architecture. The **MMU** consists of 16 independent vector processors and a reconfigurable adder network to accommodate different partial sum accumulation requirements arising from diverse dataflows. Each vector processor features a MAC line capable of performing 64 MAC operations in a single cycle and a dedicated local buffer to store operands and partial sums. The **divider unit** employs a reciprocal-based single-divisor approach. Since all divisions in transformers fall under the single-divisor category (scaling, Softmax, normalization), a single reciprocal generator is employed for efficient calculation, distributing the result to 64 different multipliers. The **exponent unit** can perform 64 parallel exponent calculations based on Taylor-series approximation of the exponent function. The **absolute distance unit** can process up to 64 different outputs simultaneously, and the **clustering unit**, equipped with 48 comparators, can process up to 16 parallel 3-input comparisons within a single cycle. This customized architecture efficiently exploits data parallelism, enabling significant performance improvements for row-clustering based attention computations.

4.3 SpARC Computation Pipeline

In SpARC, data dependencies inherent in the computational flow should be carefully considered to execute the row-clustering approximate attention of Section 3 efficiently. As illustrated in Fig. 3, the centroid initialization, similarity computation, and cluster allocation appear to be performed sequentially. However, many row-wise computations are independent of each other, allowing for parallel processing of rows. This means that once a row of $Q\hat{K}^T$ is processed, it can be forwarded to the centroid initialization step while the remaining rows of $Q\hat{K}^T$ are still being calculated. Exploiting this row-wise data independence for increased throughput, the SpARC architecture employs a row-stationary dataflow, enabling parallel processing across different computational stages.

Fig. 5 illustrates the computation pipeline of the SpARC architecture. In the initial stage, the MMU computes the full matrices of Q

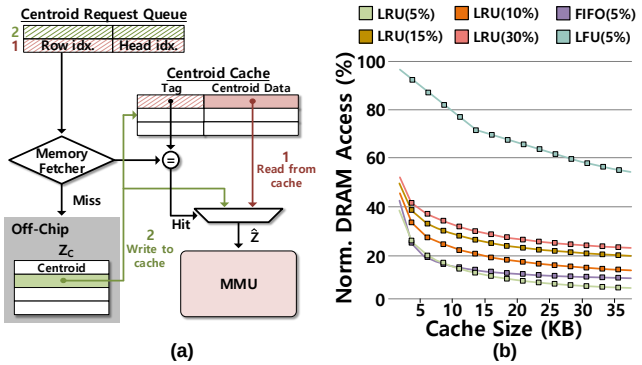

Figure 6: (a) Centroid cache architecture (b) Normalized DRAM access versus different cache sizes for multiple replacement policies and attention map density

and K. Once the Q and K matrices are obtained, the MMU proceeds with the row-wise computation of $Q\hat{K}^T$. As each row of $Q\hat{K}^T$ is computed, it is forwarded to the adder network and divider units for interpolated centroid initialization. Similarly, each computed centroid is then passed to the divider unit and absolute distance unit for normalization and L1 distance computation.

Upon completing each row of the similarity matrix, the index searcher within the clustering unit begins traversing the matrix to allocate each row of $Q\hat{K}^T$ to a cluster and update the corresponding centroid. While the L1 distance and cluster allocation are being computed, the MMU concurrently calculates the V matrix, ensuring high throughput by eliminating MMU idle time. Once the generation of V and cluster allocation is complete, the corresponding rows of each centroid are gathered to perform the low-cost Q_cK^T, followed by Softmax, $Z_c = A_cV$, and linear projection computations.

By parallelizing various computations without data dependencies on distinct specialized sub-processors, we effectively minimize the overhead associated with the clustering process, thereby establishing matrix multiplication as the critical path within the self-attention mechanism.

4.4 Cluster Centroid Cache

Following the completion of $Z_c = A_cV$ computation and multi-head self-attention concatenation, the calculated centroid results of $\mathbb{R}^{C\times D}$ should be transformed to back to the original input dimension of $\mathbb{R}^{N\times D}$ to facilitate subsequent computations, including linear projection and feed-forward network. This process involves replacing the rows in each cluster with their corresponding centroids, as previously described in Section 3.2 Cloning. While the proposed algorithm and hardware architecture significantly improve energy efficiency and speed, the dynamic nature of the clustering process and the irregular allocations of rows to clusters introduce main memory overheads due to unpredictable access patterns. Loading centroids from main memory upon each request leads to fragmented instead of contiguous memory access, resulting in excessive data movements, reduced memory bandwidth utilization, and increased memory access energy consumption.

Mitigating this issue, we propose incorporating an additional memory hierarchy called centroid cache. As illustrated in Fig. 6 (a), instead of explicitly cloning and storing centroids in main memory, the request is sent to the memory fetcher when a centroid is requested for computation. The fetcher checks the cache to determine if the requested centroid is present. If a cache hit occurs (request 1), the fetcher retrieves the centroid from the cache, eliminating the need to access main memory. In case of a cache miss (request 2), the fetcher loads the centroid from the main memory, storing it in the centroid cache for future use.

Fig. 6 (b) demonstrates the reduction in main memory access bits in relation to cache size for various attention map densities and replacement policies. The data reveals a saturation point for memory access reduction, leading to the selection of a 32KB centroid cache size. Additionally, considering the data access pattern locality of the required centroids, we employ the least recently used (LRU) replacement policy instead of least frequently used (LFU) or first-in-first-out (FIFO) policies to further optimize memory access. This additional memory hierarchy effectively reduces main memory access to approximately 20%.

Table 1: Algorithm result

		Density	F1 Score	Exact Match
SQuAD v1.1 BERT-Base	Baseline	100%	87.34%	79.78%
	Sanger	17.30%	86.59%	78.59%
	Ours	**15.88%**	**87.01%**	**79.39%**
		29.89%	**87.28%**	**79.52%**
		Density	Matched Acc.	
MNLI BERT-Base	Baseline	100%	83.91%	
	Sanger	18.81%	83.52%	
	Ours	**15.90%**	**83.55%**	
		28.11%	**83.62%**	
		Density	Acc.	
CLOTH BERT-Base	Baseline	100%	79.34%	
	Sanger	18.30%	78.90%	
	Ours	**16.84%**	**78.94%**	
		29.89%	**79.19%**	
		Density	Perplexity	
Wikitext-2 GPT-2	Baseline	100%	21.3	
	Sanger	16.27%	21.52	
	Ours	**16.37%**	**21.5**	
		29.28%	**21.34**	
		Density	Acc.	
ImageNet-1K ViT-Base	Baseline	100%	84.40%	
	Sanger	-	-	
	Ours	**15.15%**	**83.61%**	
		26.42%	**84.28%**	

5 EXPERIMENTAL RESULTS

5.1 Methodology

To validate the effectiveness of the proposed row-clustering approximate attention, we evaluate it on three different transformer models across five distinct tasks: BERT-Base [2] on SQuAD v1.1, MNLI, CLOTH, GPT-2 [3] on Wikitext-2, and ViT [5] on ImageNet-1K. For error profiling and low-rank extraction, 1K random samples of the training dataset were used, and each transformer model was fine-tuned after applying the proposed algorithm. In the low-rank extraction process, 15% of columns of K are extracted.

The proposed SpARC architecture is implemented in RTL and synthesized using Synopsis Design Compiler under 28nm CMOS technology. The power and energy consumption are obtained using

Table 2: Hardware configuration

Module	Parameter	Area (mm^2)	Power (mW)
Core	64×16 INT16	1.676	322.39
Memory, NoC	1MB (32KB cache)	1.556	697.60
Exp. Unit	64 Exp.	0.081	19.56
Div. Unit	$1/n$ based 64 INT32	0.232	98.18
Clust. Unit	48 Comparator	0.272	108.97
Dist. Unit	64 INT32	0.014	5.62
Total		$3.83mm^2$	1.252W
Operating Freq.		500MHz	

PrimeTime PX. The speedup is obtained with an in-house developed cycle-accurate simulator with HBM2 as main memory [13]. To prove the effectiveness of SpARC, we compare it with two general-purpose computing platforms: EdgeCPU, EdgeGPU, and two SOTA sparse transformer accelerators: SpAtten [14] and Sanger [15]. The EdgeCPU and EdgeGPU are both from NVIDIA Jetson Nano. SpARC, SpAtten, and Sanger are evaluated with 15% attention map density for fair comparisons.

5.2 Algorithm Evaluation

Table 1 presents the experimental results for each task. We compare our approach against Sanger [15], a state-of-the-art sparse transformer accelerator. For BERT and GPT-2, we achieve an attention map density of approximately 15% with negligible accuracy loss across all five tasks. Notably, our work outperforms Sanger on SQuAD, CLOTH, and Wikitext-2 while exhibiting slightly lower performance on MNLI. This discrepancy is potentially due to MNLI's considerably smaller maximum sequence length, aligning better with Sanger's highly unstructured sparsity map.

Regarding FLOPs reduction and cluster number, our proposed approach significantly reduces computational complexity compared to the dense baseline. The number of clusters with an attention map density of ~15% translates to a FLOPs reduction of approximately 82%, indicating a minor computational overhead of the low-rank approximation process. This reduction in FLOPs directly contributes to the speedup gains, enabling faster and more energy-efficient transformer processing.

5.3 Hardware Evaluation

Table 2 shows the synthesized results of SpARC. The area of SpARC is $3.83mm^2$, and power is 1.252W with an operating frequency of 500MHz, achieving a peak energy efficiency of 408.95 GOPs/W. Fig. 7 (a) shows the core attention speedup of SpARC against various hardware baselines. SpARC consistently outperforms all other baseline hardware on all different benchmarks. On average, SpARC achieves a speedup of 147.8×, 10.6×, 3.1×, and 2.7× over EdgeCPU, EdgeGPU, SpAtten, and Sanger, respectively. Considering the energy efficiency of Fig. 7 (b), SpARC achieves an end-to-end energy efficiency of 531.4×, 36.5×, 4.6×, and 3.1× over EdgeCPU, EdgeGPU, SpAtten, and Sanger, respectively.

6 CONCLUSION

In this paper, we introduce SpARC, a novel sparse transformer accelerator that employs row-wise clustering-based approximate attention and data-independent parallel processing to achieve substantial performance gains. By effectively reducing self-attention computational complexity and the number of memory accesses, SpARC enables the efficient execution of transformers while maintaining accuracy. Experimental results demonstrate that SpARC surpasses the performance of state-of-the-art sparse transformer accelerators across various tasks and transformer models.

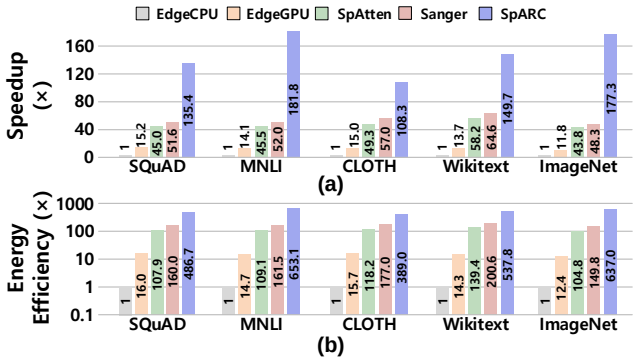

Figure 7: (a) Normalized core attention speedup (b) Normalized end-to-end energy efficiency

7 ACKNOWLEDGMENTS

This work was supported by the National Research Foundation of Korea grant funded by the Korean government (No. NRF2022M3I7A2079267); in part by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No. 2022-0-00266, Development of Ultra-Low Power Low-Bit Precision Mixed-Mode SRAM PIM); in part by the Samsung Research Funding and Incubation Center of Samsung Electronics under Grant SRFCMA1802-01. The EDA tool was supported by the IC Design Education Center (IDEC), Korea.

REFERENCES

[1] Ashish Vaswani et al. Attention is all you need. In *Proc. of NeurIPS*, volume 30, pages 6000–6010, 2017.
[2] Jacob Devlin et al. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proc. of NAACL-HLT*, pages 4171–4186, 2019.
[3] Alec Radford et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.
[4] Tom Brown et al. Language models are few-shot learners. In *Proc. of NeurIPS*, volume 33, pages 1877–1901, 2020.
[5] Alexey Dosovitskiy et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*, 2020.
[6] Nikita Kitaev et al. Reformer: The efficient transformer. In *ICLR*, 2019.
[7] Aurko Roy et al. Efficient content-based sparse attention with routing transformers. *TACL*, 9:53–68, 2021.
[8] Gonçalo M Correia et al. Adaptively sparse transformers. In *Proc. of EMNLP-IJCNLP*, pages 2174–2184, 2019.
[9] Baiyun Cui et al. Fine-tune bert with sparse self-attention mechanism. In *Proc. of EMNLP-IJCNLP*, pages 3548–3553, 2019.
[10] Yi Tay et al. Sparse sinkhorn attention. In *Proc. of ICML*, pages 9438–9447. PMLR, 2020.
[11] Eric Qin et al. Sigma: A sparse and irregular gemm accelerator with flexible interconnects for dnn training. In *Proc. of HPCA*, pages 58–70. IEEE, 2020.
[12] Radhakrishna Achanta et al. Slic superpixels compared to state-of-the-art superpixel methods. *TPAMI*, 34(11):2274–2282, 2012.
[13] Mike O'Connor et al. Fine-grained dram: Energy-efficient dram for extreme bandwidth systems. In *Proc. of MICRO*, pages 41–54, 2017.
[14] Hanrui Wang et al. Spatten: Efficient sparse attention architecture with cascade token and head pruning. In *Proc. of HPCA*, pages 97–110. IEEE, 2021.
[15] Liqiang Lu et al. Sanger: A co-design framework for enabling sparse attention using reconfigurable architecture. In *Proc. of MICRO*, pages 977–991, 2021.