A 40nm 5.6TOPS/W 239GOPS/mm^2 Self-Attention Processor with Sign Random Projection-based Approximation

Seong Hoon Seo*, Soosung Kim*, Sung Jun Jung, Sangwoo Kwon, Hyunseung Lee, Jae W. Lee
Seoul National University, Seoul 08826, Republic of Korea
{andyseo247, soosungkim, miguel92, kwonsw055, hs_lee, jaewlee}@snu.ac.kr

***Abstract*—Transformer architecture is one of the most remarkable recent breakthroughs in neural networks, achieving state-of-the-art (SOTA) performance on various natural language processing (NLP) and computer vision tasks. *Self-attention* is the key enabling operation for transformer-based models. However, its quadratic computational complexity to the sequence length makes this operation the major performance bottleneck for those models. Thus, we propose a novel self-attention accelerator that skips most of the computation by utilizing an approximate candidate selection algorithm. Implemented in a 40nm CMOS technology, our 5.64 mm^2 chip operates at 100-600 MHz consuming 48.3-685 mW to achieve the energy and area efficiency of 0.354-5.61 TOPS/W and 239 GOPS/mm^2, respectively.**

I. Introduction

Since the advent of the transformer architecture [1], the self-attention mechanism has become one of the most important building blocks in neural networks (NN). From the representative NLP models such as BERT [2] and GPT-3 [3] to computer vision [4] and even multi-modal models [5], the self-attention mechanism has widely been adopted in various domains, and its importance will continue to grow.

The self-attention mechanism is an operation that computes the relations among input entities. Input entities may vary depending on the context. For instance, an entity is a word or token in NLP, while it may be an image patch in vision classification. Fig. 1 is a visualization of one self-attention operation within BERT [6]. The darker the connected line is, the stronger the relationship between the two words is. In the example, the adverb "where" strongly attends to "in" and "Milan", while the pronoun "it" attends to "conference".

Convolutional and recurrent operations, which are the dominant primitives for vision and NLP, respectively, only capture the relationship among local, adjacent entities [7]. Thus, they cannot directly compute the global dependencies. Self-attention mechanism overcomes this limitation and directly captures the long-range dependencies among distant entities.

$$\text{Self-attention}(Q, K, V) = \text{Softmax}(\frac{QK^T}{\sqrt{d}})V \qquad (1)$$

Equation (1) shows the formal definition of the self-attention operation. It takes $n \times d$ query (Q), key (K), and value (V) matrices as inputs, which are linear projections of the input entities. n is the sequence length of the input entities whose attention will be computed, and d is the dimension of projected vectors for each entity. QK^T is often divided by $\sqrt{d}$ as a scaling factor for stable gradients [1]. The softmax function ($\sigma(V)_i = \frac{e^{V_i}}{\sum_{j=1}^{n} e^{V_j}}$) returns a weight, which is then multiplied by V to output a weighted sum of value vectors.

* These authors contributed equally to this work.

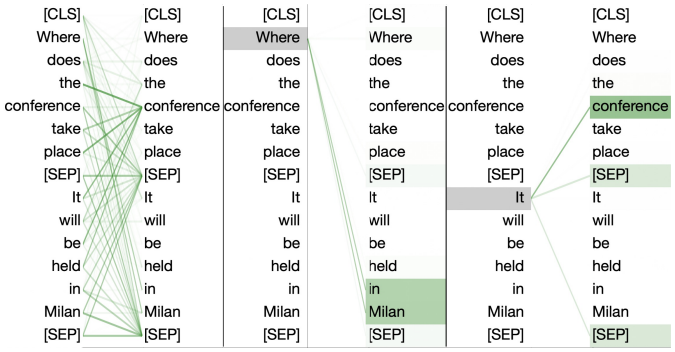

Fig. 1. Visualization of the self-attention mechanism using BERT [6]

Although the self-attention operation is mainly composed of matrix multiplications that benefit from the parallelism of GPUs and NPUs, it is computationally expensive. The amount of required computation quadratically increases as the length of the input entities (i.e., n) increases, as the self-attention mechanism has a complexity of $O(n^2d)$. This often limits the maximum length of the input sequence to 512 or less for BERT. Therefore, input sentences must be split into multiple sub-sequences, preventing the model from capturing the global relationship between input entities in different sub-sequences.

A recent study shows that the self-attention mechanism can be *approximated* with negligible accuracy loss by estimating the angular distance between a query-key vector pair based on sign random projection (SRP) [8]. From the observation that the softmax function maps most of the values to near-zero except for a few large values, Ham et al. [8] proposes an approximation algorithm that prunes most query-key pairs that do not contribute to the final output.

We elaborate on this idea, presenting a 40nm implementation of a self-attention accelerator with SRP-based approximation. The implemented chip is fully integrated with a popular NN framework for end-to-end inference of NLP models. Our chip achieves a peak energy efficiency of 5.61 TOPS/W, 202× higher than a high-performance GPU (NVIDIA GV100), and the SOTA technology-normalized area efficiency.

The remaining paper is organized as follows. Section II overviews the SRP-based approximation in [8], followed by implementation details of the chip in Section III. Section IV presents experimental results, and Section V concludes.

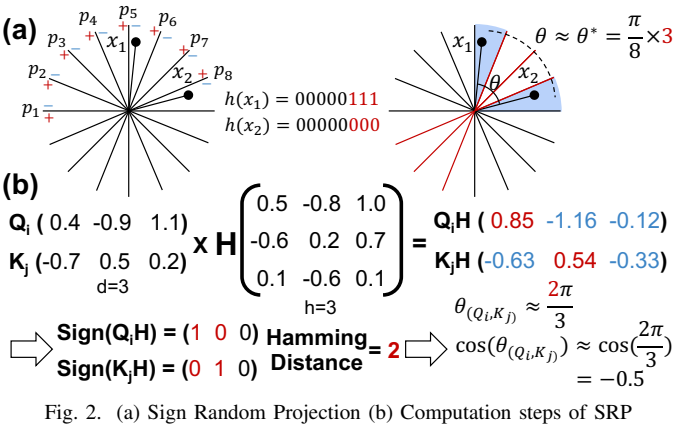

Fig. 2. (a) Sign Random Projection (b) Computation steps of SRP

II. Sign Random Projection-based Approximation

This section summarizes the approximation algorithm that utilizes sign random projection (SRP) to estimate the dot-product similarity [8]. The first stage of the self-attention mechanism is the multiplication of query (Q) and transposed key (K^T) matrix. This matrix multiplication computes the dot-product similarity between each pair of d-dimensional query vector (Q_i) and key vector (K_j), where ith query/key vector is a linear projection of the ith input entity. In other words, $QK^T_{(i,j)} = Q_i \cdot K_j$.

The softmax function sharpens the difference between values to project most values to near-zero except for a few large ones, which also make near-zero the multiplication of such values by the value matrix (V) (i.e., Softmax$(QK^T)_{(i,j)} \cdot V_{(j,d)}$), to barely affect the final outcome. Based on this observation, Ham et al. [8] proposes a method to identify and skip (Q_i, K_j) pairs whose softmax output will be near-zero, significantly reducing the amount of computation without loss of accuracy.

Based on $(Q_i \cdot K_j)/||Q_i|| = ||K_j||cos\theta$, the scheme approximates the magnitude of the dot-product similarity by computing the norm of key vectors and the estimated angle in advance. As illustrated in Fig. 2(a), SRP estimates the angle between two vectors with the Hamming distance between hashed bit-vectors. Binary hashing is performed using the relative location against each hyperplane (p_i). Fig. 2(b) shows how SRP is computed. Each input vector is first multiplied with a $d \times h$ hash matrix (H), where each column vector corresponds to a different hyperplane. The output is then transformed into a h-bit vector based on the sign of each value. The Hamming distance of two bit-vectors estimates the angle.

Using the estimated angle, The query-normalized similarity ($||K_j||cos\theta$) can be estimated, which is then compared against the product of $\max_{1 \le j \le n} ||K_j||$ and pre-trained cosine threshold $cos\theta_t$. Query-key vector pairs whose estimated similarity is below this threshold are skipped. $\cos\theta_t$ is computed for each value of the configurable hyperparameter p using the training dataset. p controls the degree of approximation, with a higher value resulting in a more aggressive approximation. For each p, we find $\cos\theta_t$ that selects query-key pairs whose softmax output is greater than p/n. We discuss how the choice of p affects the accuracy and latency in Section IV.

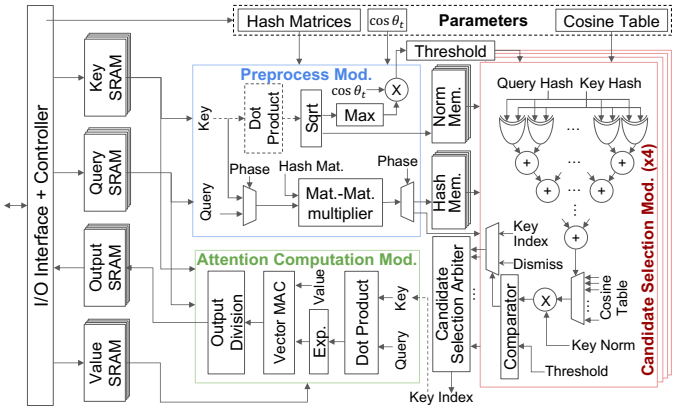

Fig. 3. Architecture of the self-attention accelerator

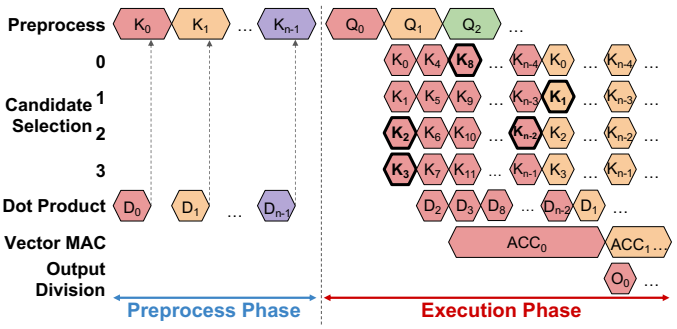

Fig. 4. Execution flow of the self-attention accelerator

III. Implementation Details

A. Overview

Fig. 3 depicts the overall architecture of the implemented accelerator. It consists of one preprocess module, four parallel candidate selection modules, one attention computation module, and a controller in charge of state transition and off-chip communication. We envision our design to be integrated into existing processors (e.g., GPU or NPU) such that the accelerator can directly access the input matrices from the device's scratchpad memory. However, for validation and measurement of our accelerator, the prototype chip includes 192 KB SRAM for input and output matrices.

Execution Flow. The execution flow is depicted in Fig. 4. During the preprocess phase, the hash and the norm of key vectors are computed by the preprocess module. During the execution phase, the same module computes the hash of each query vector. Candidate selection modules consume the query hash and the preprocessed values to deliver the indices of selected query-key pairs (e.g., $(Q_0, K_2), (Q_0, K_3)$ in Fig. 4) to the attention computation module, which in turn performs the self-attention operation in a pipelined manner.

Number Representations. The proposed accelerator uses INT12 and custom FP17 with a 6-bit mantissa. For dot-product similarity, INT12 is precise enough to prevent the loss of end-to-end accuracy of the model. For exponent and its multiplication with the value matrix, custom FP17 with 10-bit exponent and 6-bit mantissa is used. This prevents the overflow of the softmax output's denominator (i.e., $\sum_{j=1}^{n} e^{V_j}$) and allows us to decompose the softmax function into the

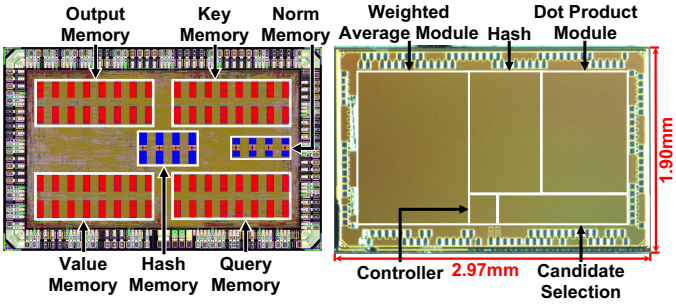

Fig. 5. Chip layout (left) and micrograph (right)

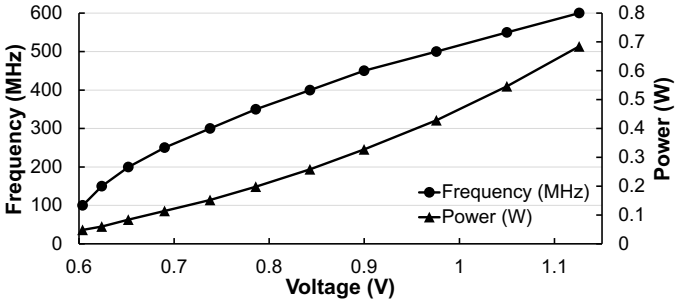

Fig. 6. Voltage-frequency scaling

accumulation stage and division stage, hiding the computation bottleneck of exponent accumulation required before division.

B. Hardware Details

Preprocess Module. For each input vector, the preprocess module computes its h-bit hash value using SRP. We choose h=d=64 for our implementation. Naïvely multiplying a $d \times h$ matrix incurs too much overhead for preprocessing. Instead, we use the algorithm in [8] to decompose one large matrix multiplication into three smaller multiplications. In h=d=64, this reduces the required number of multiply-and-accumulate (MAC) operations per hashing from 64^2=4096 to $3 \cdot 4^4$=768.

During the preprocess phase, the module also computes the L2 norm of each key vector, keeping track of the maximum key norm. As mentioned in Section II, the maximum key norm is multiplied by a pre-trained cosine threshold to generate a similarity threshold. Norm is computed by taking the dot product of the input vector itself and taking the square root of the product. Since the attention computation module is idle during this phase, the preprocess module "borrows" its dot-product unit, maximizing resource utilization.

Candidate Selection Module. Four candidate selection modules operate in parallel during the execution phase. Query hash is passed from the preprocess module, and the key hashes are retrieved from the hash memory. The module first computes the Hamming distance between the query and key hashes. A lookup table maps the result to the corresponding cosine value, which is multiplied by the key norm to compute query-normalized similarity. Finally, the module compares the similarity against the threshold and passes the indices of selected query-key pairs to the attention computation module.

Attention Computation Module. Attention computation module is composed of a dot-product unit and a weighted average sub-module. The former computes the dot product of query-key vector pairs selected by candidate selection modules. The latter consists of an exponent, a vectorized MAC, and an output division unit. These units are pipelined to compute the weighted sum of value vectors. First, the exponent unit returns the exponent of the dot product (e^{D_i}), internally accumulating each exponent. Second, the vectorized MAC unit multiplies the exponent with the value vector, then accumulates the output. Once accumulation for a single query vector is complete, the output division unit divides the accumulated vector with the sum of exponents ($\sum e^{D_i}$). As shown in Fig. 4, performing division at the end of the pipeline prevents unnecessary stalls waiting for the sum of exponents.

IV. Measurement Results

The prototype chip shown in Fig. 5 is fabricated in 40nm CMOS technology with an area of 5.64 mm^2. As shown in Fig. 6, the chip operates at a maximum frequency of 600 MHz with 1.126 V.

Methodology. We measure the performance of the chip on BERT-large model [2] using Stanford Question Answering Dataset (SQuAD 2.0) [9]. To compare the performance with existing processors, we run the same workload on NVIDIA GV100 GPU [10], Jetson TX2 GPU [11], and Google TPUv3 [12]. The performance of these processors is measured using FP16/BF16, advantageous for them compared to using FP32.

We develop an evaluation environment that allows the end-to-end inference of BERT on the existing ML framework (PyTorch) with offloading of the self-attention mechanism to the chip. Fig. 7 depicts the details of the offloading mechanism. During inference, the self-attention operations are exported to the C++ offloader, which exposes an API for the end-user. The offloader forwards the input matrices to the PCIe-connected FPGA, which relays them to the chip. Once the chip completes the computation, the FPGA receives the output matrix from the chip, relaying it back to the offloader.

Performance. The chip achieves peak energy and area efficiency of 5.61 TOPS/W and 239 GOPS/mm^2 as shown in Table I. The chip is up to 92.5× more energy-efficient than TPUv3, which has the highest TOPS/W among the three. Fig. 8 shows the effect of approximation on performance and accuracy. As the degree of approximation increases, the chip aggressively skips more query-key vector pairs, directly leading to throughput improvement. Moderate approximation of p=2 achieves 3.4× speedup over no approximation (p=0), with negligible (sub-1%) accuracy loss compared to the baseline

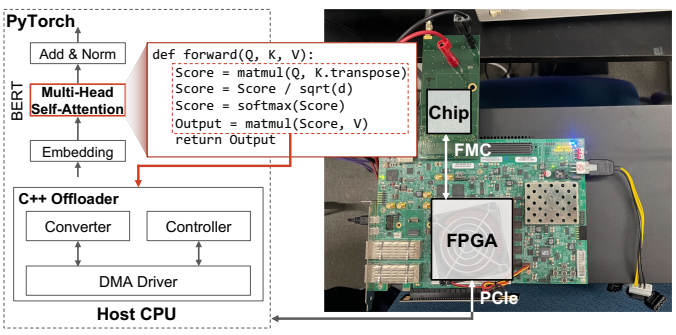

Fig. 7. End-to-end inference system based on PyTorch

TABLE I. COMPARISON TABLE

	GV100 GPU [10]	Jetson TX2 GPU [11]	TPU v3 (v3-8) [12]	ISSCC 2022 [13] [a]	This Work
Technology (nm)	12	16	16	28	40
Die Area (mm^2)	815	31.4[b]	700	6.82	5.64
Frequency (MHz)	1627	1300	940	50 - 510	100 - 600
Power (W)	250	8.69	1800	0.012 - 0.273	0.0483 - 0.685
ML Algorithm	BERT-large	BERT-large	BERT-large	GPT-2, ViT, Swin-Transformer	BERT-large
ML Dataset	SQuAD 2.0	SQuAD 2.0	SQuAD 2.0	wikiText-2, ImageNet	SQuAD 2.0
Peak Performance (TFLOPS, TOPS)	6.95	0.14	109	1.92[c]	1.35[d]
Energy Efficiency (TFLOPS/W, TOPS/W)	0.0278 (0.00250)[e]	0.0163 (0.00260)[e]	0.0607 (0.00971)[e]	2.60 - 17.2[c] (1.27 - 8.41[c])[e]	0.354 - 5.61[d]
Area Efficiency ($GFLOPS/mm^2$, $GOPS/mm^2$)	8.53 (0.768)[e]	4.50 (0.720)[e]	156 (25.0)[e]	282[c] (138[c])[e]	239[d]

a) The effect of computations outside the self-attention layers are excluded as explained in Section IV b) Die area is approximated with the number of shader cores
c) 90% output sparsity d) 88% output sparsity e) Assumes 40nm implementation and energy is proportional to (Technology)^2 [14]

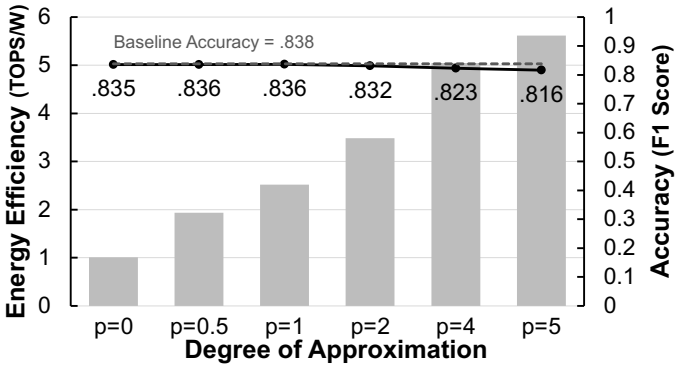

Fig. 8. Effect of approximation on performance and accuracy

accuracy of GPU using FP32 (0.838 in Fig. 8). If further accuracy loss can be traded in for increased performance, a more aggressive approximation (p=5) can lead to over 5.5× speedup at the cost of a small (some 2%) accuracy loss.

Comparison with transformer accelerator by Wang et al. [13]. Table I compares our prototype chip with a recently proposed transformer accelerator [13]. For fair comparison, dense feed-forward layers are removed as they are out of scope of our design and also lower the transformer accelerator's TOPS and TOPS/W. Peak TOPS and TOPS/W are measured at 88% output sparsity (p=5) for our chip and 90% for the accelerator by Wang et al. In contrast, minimum TOPS and TOPS/W are measured with no approximation in both cases.

Our chip results in 0.354-5.61 TOPS/W, 239 $GOPS/mm^2$. By comparison, the transformer accelerator achieved 1.27-8.41 TOPS/W, 138 $GOPS/mm^2$ on average, after normalizing the technology based on the simple scaling model presented by Biswas and Chandrakasan [14], where both energy and area are proportional to the square of technology. Our self-attention accelerator achieves comparable energy efficiency and higher area efficiency against the transformer accelerator due to the absolute computation reduction of our candidate selection mechanism.

V. CONCLUSION

This paper presents a self-attention accelerator with an approximate candidate selection algorithm. This chip is fabricated in 40nm CMOS with an area of 5.64 mm^2. Our chip selects query-key vector pairs that are estimated to be highly related, skipping the computation for the non-selected pairs. Thanks to the efficient candidate selection algorithm, the chip achieves 92.5× higher peak TOPS/W than TPUv3 and 1.73× higher technology-normalized area efficiency than the state-of-the-art transformer accelerator.

ACKNOWLEDGMENTS

This work was supported by a National Research Foundation of Korea (NRF) grant (NRF-2020R1A2C3010663) and an Institute of Information & Communications Technology Planning & Evaluation (IITP) grant (2021-0-00105), funded by the Korea government (MSIT). The EDA tool was supported by the IC Design Education Center (IDEC), Korea. Jae W. Lee is the corresponding author.

REFERENCES

[1] A. Vaswani *et al.*, "Attention is All you Need," in *NeurIPS*, 2017.
[2] J. Devlin *et al.*, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in *NAACL*, 2019.
[3] T. Brown *et al.*, "Language Models are Few-Shot Learners," in *NeurIPS*, 2020.
[4] A. Dosovitskiy *et al.*, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale," in *ICLR*, 2021.
[5] W. Kim *et al.*, "ViLT: Vision-and-Language Transformer Without Convolution or Region Supervision," in *ICML*, 2021.
[6] J. Vig, "A Multiscale Visualization of Attention in the Transformer Model," in *ACL*, 2019.
[7] X. Wang *et al.*, "Non-Local Neural Networks," in *CVPR*, 2018.
[8] T. J. Ham *et al.*, "ELSA: Hardware-Software Co-design for Efficient, Lightweight Self-Attention Mechanism in Neural Networks," in *ISCA*, 2021.
[9] P. Rajpurkar *et al.*, "Know What You Don't Know: Unanswerable Questions for SQuAD," in *ACL*, 2018.
[10] "Quadro GV100," https://www.nvidia.com/en-us/design-visualization/store/, NVIDIA, 2018.
[11] "Jetson TX2 Module," https://developer.nvidia.com/embedded/jetson-tx2, NVIDIA, 2017.
[12] N. P. Jouppi *et al.*, "Ten Lessons From Three Generations Shaped Google's TPUv4i," in *ISCA*, 2021.
[13] Y. Wang *et al.*, "A 28nm 27.5TOPS/W Approximate-Computing-Based Transformer Processor with Asymptotic Sparsity Speculating and Out-of-Order Computing," in *ISSCC*, 2022.
[14] A. Biswas *et al.*, "CONV-SRAM: an energy-efficient SRAM with in-memory dot-product computation for low-power convolutional neural networks," in *JSSC*, vol. 54, no. 1, 2019.