In-Memory Computing based Accelerator for Transformer Networks for Long Sequences

Ann Franchesca Laguna*, Arman Kazemi †, Michael Niemier ‡, X. Sharon Hu §
Department of Computer Science and Engineering,
University of Notre Dame,
Notre Dame, Indiana, USA
Email: *alaguna@nd.edu, †akazemi@nd.edu, ‡mniemier@nd.edu, §shu@nd.edu

***Abstract*—Transformer networks have outperformed recurrent neural networks and convolutional neural networks in various sequential tasks. However, scaling transformer networks for long sequences has been challenging because of memory and compute bottlenecks. Transformer networks are impeded by memory bandwidth limitations because of their low operation per byte ratio resulting in low utilization of GPU's computing resources. In-memory processing can mitigate memory bottlenecks by eliminating the transfer time between memory and compute units. Furthermore, transformer networks use neural attention mechanisms to characterize the relationships between sequence elements. Efficient hardware solutions have been proposed to implement efficient attention mechanisms, which include ternary content addressable memories (TCAM), crossbar arrays (XBars), and processing in-memory (PIM). However, these solutions do not implement a multi-head self-attention mechanism. We propose using a combination of XBars and CAMs to accelerate transformer networks. We improve the speed of transformer networks by (1) computing in-memory, thus minimizing the memory transfer overhead, (2) caching reusable parameters to reduce the number of operations, (3) exploiting the available parallelism in the attention mechanism, and (4) using locality sensitive hashing to filter the number of sequence elements by their importance. Our approach achieves a 200x speedup and 41x energy improvement for a sequence length of 4098.**

***Index Terms*—Transformers, crossbars, TCAM, LSH, parallelization, in-Memory Computing**

I. Introduction

Transformer networks [1] have gained popularity in recent years because of their ability to outperform recurrent neural networks (RNN) and convolutional neural networks (CNN) in various sequence-based task. Various transformer networks, such as BERT [2], ALBERT [3], Megatron [4], GPT3 [5], and XLNet [6] holds the best accuracies in various natural language processing applications such as machine translation, named entity recognition, and question answering. Bigger transformer networks have been designed to achieve higher accuracies in multiple tasks but require millions of parameters [4], [5]. Increasing the length of a sequence (i.e., sentences, paragraphs) also increases these parameters quadratically. As the model size and sequence length increase [3], the communication overhead needed to transfer parameters from the memory to the compute unit becomes a significant bottleneck.

Various software solutions aim to reduce the number of parameters through model parallelization [4], cross-layer parameter sharing [3], hashing [7], quantization, and pruning [8], [9]. However, these solutions are still constrained by the memory bandwidth because transformer networks do not fully utilize the compute units and have a low arithmetic intensity (FLOPS/byte). For example, Megatron [4], one of the largest transformer networks to date, only achieves 30% of the theoretical peak FLOPS of GPUs. Processing-in-memory (PIM) can reduce the memory-compute communication overhead.

Transformer networks are based mostly on neural attention mechanisms that determine the relationships between elements of a sequence. Neural attention mechanisms focus on or *attend* to the most relevant elements of the sequence for a given task. Transformer networks use multi-head attention (MHA) that performs scaled-dot product (SDP) attention in different attention heads, where each head represents a different subspace. The SDP-MHA attention can achieve a complexity of $O(1)$ with sufficient parallelization. However, the complexity of SDP-MHA in the GPU is limited to $O(dn^2/c)$, where n is the sequence length, d is the number of feature embedding dimensions, and c is the number of parallel cores. It is also limited by the memory bandwidth as memory requirements increase quadratically with respect to n.

Hardware solutions implementing attention mechanisms use ternary content addressable memories (TCAMs) [10], [11], crossbar arrays (XBars) [12], [13], and PIM [14]. TCAMs are associative memories that can store '0', '1', and don't care ('x') states and are used for attention-based computations using a similarity function (e.g., L_1, L_∞ distance [10], [12], [15] and locality sensitive hashing (LSH) [11]). XBars are circuits that can implement highly parallel matrix-vector multiplications (MVMs) and can be used to implement dot products and a modified cosine distance [13], while PIM can compute L_1 distance-based attention [12], [14]. However, these hardware implementations focused on recurrent and memory augmented neural networks and do not tackle the SDP-MHA attention needed for transformer networks. Transformer Network SDP-MHA have more parallelization opportunities than normal attention mechanisms and the number of memory entries vary with respect to the sequence length. The SDP attention is most amenable to XBars. However, as the sequence length increases, the energy consumption can increase dramatically. LSH, an approximate nearest neighbor technique, gauges the closeness between points using Hamming distances. These Hamming distances, however, cannot be used as attention scores. Hence, SDP should be computed after LSH, albeit with fewer entries.

We propose iMCAT, an in-Memory Computing-based Accelerator for transformer network inference, that (1) employs a combination of XBars and CAMs to execute scaled dot-product multi-head attention (SDP-MHA), (2) uses the XBars and TCAMs not only to store weights, but also as embedding caches to reduce the number of computations, (3) parallelizes the attention mechanism via data duplication, and (4) employs LSH to attend only to the most relevant elements of the sequence. iMCAT achieves a 200x end-to-end delay and 41x energy improvement for bidirectional transformers at a sequence length of 4096 compared to the Titan RTX GPU.

II. Background

A. Transformer Networks

Transformer networks [1] are sequence transduction models that use attention mechanisms on an encoder-decoder architecture. A transformer network uses encoders to transform an input sequence $\mathbf{x} = (x_1, x_2, ...x_n)$ to a numeric vector representation $\mathbf{z}$. The vector $\mathbf{z}$, representing the $\mathbf{x}$ in the encoder feature space, is then transformed by the decoder to an output sequence $\mathbf{y} = (y_1, y_2, ...y_m)$. The vanilla (original) transformer design [1] uses a stack of six encoder layers and six decoder layers. Different transformer networks have varying model sizes depending on the number of layers. For example, the smallest GPT3 [5] model has 12 layers, and the largest GPT3 model has 96 layers. Some transformer designs, such as BERT [2], [3], only use encoder layers. The output vector representation $\mathbf{z}$ is decoded using a simpler neural network such as a single fully connected layer to perform classification, regression, and other similar tasks. The encoder and decoder layers are composed primarily of attention layers, which are explained in detail in Section II-B.

Transformer networks can be both memory-bandwidth limited and computation limited. GPU implementations are often bounded by the memory, particularly with longer sequences, because of the $O(dn + dn^2)$ spatial complexity, where d represents the feature embedding dimension and n is the sequence length. By computing in-memory, the memory bandwidth limitation can be alleviated. A transformer network can also be computation-limited because of the $O(dn^2)$ serialized time complexity of the multi-head attention. The $O(dn^2)$ complexity comes from each time step (sequence element) attending to every other time step (sequence element), where d represents the number of feature embedding dimensions. However, with adequate parallelism, an $O(1)$ time complexity can be achieved.

B. Attention

Attention is an important aspect of human intelligence. It allows us to determine the most important parts of a text or an image and remove (or pay less attention to) irrelevant parts. For example, our visual system can focus on a certain object in a given picture instead of processing the entire picture. Neural attention mechanisms work similarly by giving more attentional weight to more important regions. In a key-value pair scenario, the input or query vector q is compared to a set of keys k using a scoring function, obtaining the attentional weights $a(k, q)$. The attention is the weighted average of the values v using the attentional weights a. The following equation summarizes the key-value pair attention:

$$attention(q, k, v) = \sum_i a(k, q_i) v_i \tag{1}$$

A multi-head self-attention [1] is used in transformer networks, which projects the input $\mathbf{x}$ to different subspaces to obtain q, k, and v vector representations. In each head, an SDP attention (Eq. 2) is used to compare the queries to a set of key-value pairs. Instead of a dot-product, another approach [7] is to use LSH. MHA and LSH are discussed below.

1) Multi-head Attention (MHA): Multiple heads, representing different subspaces, are used to attend to multiple sequence elements simultaneously; this is called multi-head attention (MHA). The i_{th} head projects the query Q', key K', and values V' by using projection matrices W_i^Q, W_i^K, W_i^V. The projected SDP inputs Q, K, and V are calculated using Eq. 2. The attention heads are concatenated and aggregated by a linear layer.

$$attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2}$$

2) Locality Sensitive Hashing (LSH) Attention: LSH is commonly used in nearest neighbor applications to find similar items quickly. This is achieved by hashing more similar items to the same binary signature. The b-th bit in the hashing signature $h_b(q) = sign(q \cdot r_b)$ is obtained by projecting the query q using the hyperplane r. The signature then has either a value of ± 1 depending on which side of the hyperplane r, q resides. This is repeated using different hyperplanes. Both the queries and keys are hashed. The query's binary signature is then compared to the keys' binary signatures using Hamming distances. LSH attention has been used in multiple attention-based neural network architectures [7]. In particular, Reformer [7], a type of transformer network that uses angular LSH during training to improve the efficiency of the transformer network for longer sequences by reducing the complexity of the SDP attention from $O(n^2)$ to $O(n \log n)$.

C. Hardware Kernels for Acceleration

1) Crossbar Arrays (XBars): XBars can perform fast, and energy-efficient matrix-vector multiplications (MVMs) [16]. XBars are comprised of programmable memory elements on the cross-points of rows and columns and can store a matrix in the memory array. To perform MVM with an XBar, an input vector is applied to the rows of the XBar, and the output of the MVM is read at the columns using analog-to-digital converters (ADCs). XBars are leveraged for accelerating attention mechanisms based on dot-products in [12], [13]. However, neither approach explores the parallelism of MHA used in transformer networks.

2) Content Addressable Memories (CAMs): CAMs are memory fabrics that can perform parallel associative searches in-memory. A CAM stores multiple words and finds the closest word to a given query based on the Hamming distance metric.

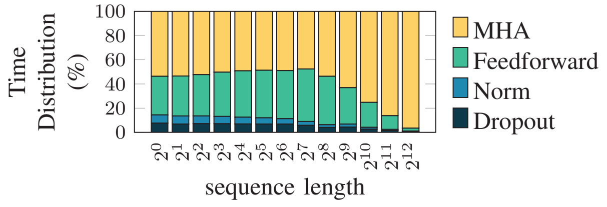

Fig. 1: Execution time distribution of the different operations in transformer networks at varying sequence lengths.

CAMs enable an LSH-based search, which has been leveraged in the literature [11] to accelerate few-shot learning tasks. A ternary CAM (TCAM) can store “0”, “1”, and “don’t care” in each cell where a cell storing “don’t care” is always a match. TCAMs have been used to implement attention using L_∞ and L_1 distance metrics through range encoding [10].

III. In-Memory Computing-Based Accelerator for Transformer Networks (iMCAT)

We introduce an accelerator framework to speed up transformer networks, particularly for long sequences. Transformer networks are composed primarily of MHA, feedforward layers, and layer normalization. As the sequence length increases, the MHA dominates the execution time, as shown in Fig. 1. Because of this, iMCAT focuses its acceleration with MHA, which relies primarily on MVMs. We use XBars to implement the majority of the operations by the MHA and the feedforward operations.

The vanilla transformer has three types of MHA: (1) a masked MHA for the decoder layer, (2) a bidirectional MHA for the encoder layer, and (3) an encoder-decoder MHA that connects the encoder to the decoder. The *masked MHA* only attends to the sequence’s past elements while the *bidirectional MHA* attends to the sequence’s past and future elements. The *encoder-decoder MHA* uses the encoder feature embeddings as keys and values and the decoder feature embeddings as queries to connect the encoder layer to the decoder layer.

In Sec. III-A, we discuss a general hardware mapping of the MHA. We then discuss the modifications to tailor-fit the general MHA to the masked, bidirectional, and encoder-decoder MHA of the transformer network in Sec. III-B. Finally, we discuss the LSH+SDP MHA that can improve the energy efficiency of iMCAT in Sec. III-C.

A. General Multihead Attention Hardware Mapping

For a general mapping (Fig. 2), the elements of the inputs, query $Q' = \{q'_t\}$, key $K' = \{k'_t\}$, and value $V' = \{v'_t\}$, arrive one at a time. As shown in Fig. 2, the MHA inputs q'_t, k'_t, and v'_t serve as inputs to multiple heads. Each head’s output is then concatenated and multiplied using the weight function W^{MHA} implemented using XBars. For the vanilla transformer, W^{MHA} requires a $d \times d$ (512×512) XBar. Each head is composed of Xbars for W^Q, W^K and W^V for the projection matrices and Xbars for K and V, serving as an attention cache.

For each head, the query, key, and value are projected to a different feature space. Each projection matrix W_i^Q, W_i^K, and W_i^V requires a $\frac{d}{h} \times d$ MVM (64×512 MVM for the vanilla transformer) implemented in Xbars. These projection matrices have static weights during inference. The projected query q_t is then compared to the projected key $k_t, k_{t-1}, k_{t-2}...$ and projected values $v_t, v_{t-1}, v_{t-2}...$ using a SDP attention. Since the projected query q_t also depends on the present and previous projected keys $k_t, k_{t-1}, k_{t-2}...$ and values $v_t, v_{t-1}, v_{t-2}...$, they can be cached in XBars for reuse to minimize the computations. The key-value XBars, hence, serve as a physical realization of the content addressable memory (different from hardware CAMs[1]) found in attention mechanisms. We use 8-bit precision, which is found to achieve good accuracies for quantized transformer networks [8], [9].

Each row of K and each column of V represent a projection k_t and v_t. K have a size of $n \times d$, and V have a size of $d \times n$. The required number of rows in the XBar arrays K increases as the sequence length increases. Since the rows are independent of each other, there are $\lceil n/64 \rceil$ XBar arrays of 64×64 as n increases. Each of the XBars for K represents 64 dimensions in an n-dimensional vector. The required number of columns of V increases as the sequence length n increases, introducing more complexity and partial sums.

To counter this complexity, we use LSH to limit the number of columns required in V. This introduces more write operations but is more beneficial as the sequence length increases. To reduce the number of computations, the keys and values for each time step are cached in the K and V XBars.

In SDP, the attention weight $a(k, q)$ is scaled by d_k. The dimension of the keys $d_k = \frac{d}{h}$, where h represents the number of heads, is limited to 2^{2b} such that *scaling* is implemented by removing b least significant bits. The common value of d_k is 64, which is equivalent to removing 3-bits. Softmax is implemented using lookup tables similar to [17].

B. Masked, Bidirectional, and Encoder-Decoder Multi-head Attention Hardware Mapping

We discuss further tailor-fiting iMCAT’s general MHA to the different types of MHAs: Encoder-Decoder MHA, Masked Multi-Head Attention, Bidirectional MHA.

1) Encoder-Decoder Attention: The encoder-decoder MHA uses keys and values from the encoder layer and the queries from the decoder layer. Once these keys and values are obtained from the encoder layer, they can be used by the decoder for each autoregressive iteration.

2) Masked Multi-Head Attention: The transformer network’s decoder layer is autoregressive by nature (the input is a delayed version of the output) and uses masked MHA. The masking

[1]Attention content-addressable memories are *software* memories that perform content-based retrieval by comparing a *floating point* query vector to a set of *floating point* keys using a similarity function such as *cosine similarity*. They are called memory, similar to a human memory, because they store information from *current* and *past* time steps. Hardware CAMs are *physical* memory fabrics that also perform content-based retrieval by comparing a *binary* query vector to the contents stored within their memory. Hardware CAMs can perform this retrieval in $O(1)$. The contents of the memory are often compared using *Hamming distance*.

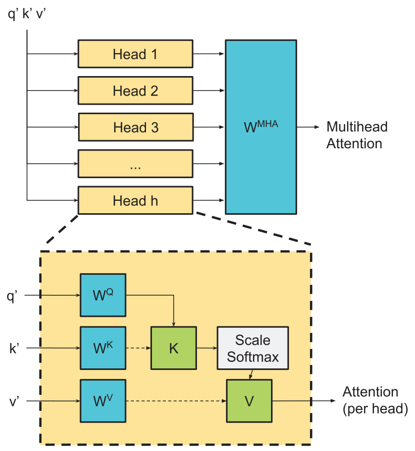

Fig. 2: General Multihead Attention Mapping where W^Q, W^K, W^V, and W^{MHA} are XBars with static weights and K and V are XBars used as attention caches.

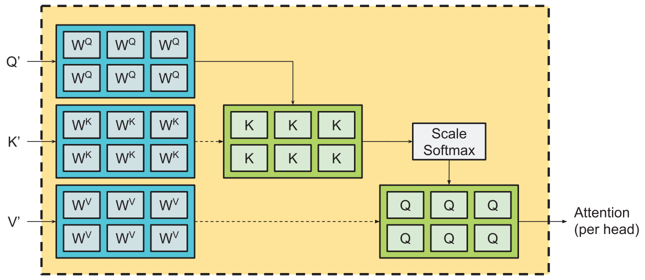

Fig. 3: To achieve parallelism the XBars are duplicated for the bidirectional MHA.

is essential to prevent a leftward information flow (the future affects the past). Masking is equivalent to only activating the rows 1 to t of the XBar for time step t. Other XBar rows are turned off to improve energy efficiency.

3) Bidirectional Multi-Head Attention: The encoder layer uses bidirectional MHA that attends to the previous, current, and future values of the input on a given time step. The bidirectional MHA can be highly parallelized as shown in Fig. 3. The MHA projection weights are duplicated by p times, where $p = 6$ in Fig. 3, to multiple XBars to provide higher parallelism. Each duplicate aims to attend to a sequence element or a timestep. If $n \leq p$, the MHA can be implemented in $O(1)$ time. If $n > p$, the complexity will be $O(n/p)$. However, increased parallelism also requires more XBars because of the duplication and thus, a higher peak power. As such, the attention parallelism is limited by the total memory size and the thermal design power. The implementation of the masked MHA is not inherently parallelizable due to the recursive nature of the decoder.

C. LSH+SDP Multihead Attention Hardware Mapping

LSH is used to reduce the number of attention computations for the XBars K and V. To implement an LSH-based attention

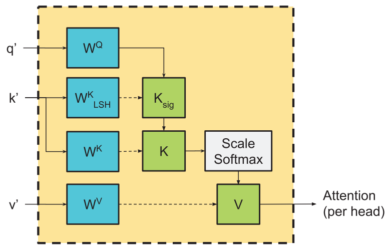

Fig. 4: LSH + SDP Attention MHA Mapping.

mechanism, the keys need to be hashed to a binary signature, where each bit in the signature is hashed using the equation $h(x) = (sign(q \cdot r) + 1)/2$. As shown in Fig. 4, instead of serializing after the keys, a XBar with weights $W^K_{LSH} = W^K R$, where R is a matrix of Gaussian distributed hyperplanes, is used to avoid additional latency as it can run in parallel with other weight matrices. Using a CAM, the binary signature of the query is compared to the signature of keys. The SDP attention is only implemented on m keys (iMCAT used $m = 16$), whose binary signatures have the lowest Hamming distance from the query's binary signature. This reduces the number of computations and improves energy consumption with a small latency overhead.

IV. Results

For comparison, we use the following for a baseline hardware: an Intel Core i7-10750H CPU (2.60GHz, 2592 Mhz, 6 cores) with a Titan RTX GPU (672 GB/sec memory bandwidth and peak performance of 130 teraFLOPS). To get the latency and energy of the baseline implementation of the transformer network, we use NVIDIA NSight. We use the parameters of the vanilla transformer [1] and BERT [2]. We use 8-bit quantization for both the GPU and the XBar-CAM implementation. 8-bit quantization has been found to be the lowest quantization for both weights and activations that achieves acceptable accuracies [8], [9].

We consider SRAM XBars implemented on a 14nm node. The energy and delay results for read and write are obtained from NeuroSim [18]. We assume a XBar size of 64×64, where each synapse has 8-bits of precision (8 SRAM cells). Each XBar has one 8-bit ADC per 8 columns, and their overhead is accounted for in the results.

We show the transformer network delay and energy improvement using iMCAT in the following sections. Sec. IV-A shows the base improvement from running MHA using XBars and caching the MHA's keys and values. Sec. IV-B shows the additional improvement of the MHA by increasing the parallelism using XBar duplication, and Sec. IV-C shows the energy improvement of using LSH followed by a dot-product for the MHA. Finally, Sec. IV-D shows the end-to-end improvement for the vanilla and BERT transformers.

A. Base Speedup and Energy Improvement

The base implementation uses the general MHA discussed in Sec. III-A using XBars. The base improvement comes from

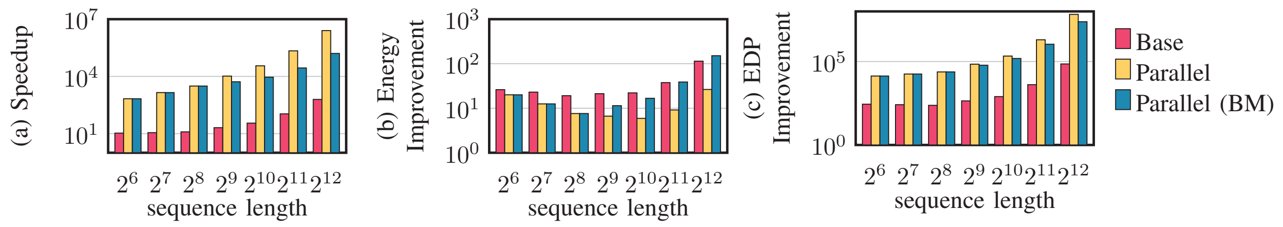

Fig. 5: (a) Speedup, (b) Energy and (c) EDP Improvement of MHA in crossbars (base), with parallelized attention and parallelized attention with bounded memory (BM)

(1) reducing memory transfer overhead, (2) caching the keys and values, and (3) using XBars for parallelism. Fig. 5 shows the improvement of implementing the MHA in the accelerator when running a bidirectional transformer for inference. The base speedup (shown as red bars in Fig. 5) of running the transformer network on the XBars at $n = 2^6 = 64$ is 10.7×. The GPU fully utilizes the memory and compute units at this sequence length and achieves optimal performance.

As the sequence length increases, the GPU's memory bandwidth and the compute limitation are reached, and the execution time starts to grow quadratically, starting at $n = 256$ by approximately $n^2/256$. By caching the keys and values and using XBars, iMCAT only grows linearly. Caching the keys and values reduce the required number of MVMs for W^Q, W^K, and W^V by n, thus reducing the delay by n. (Since the XBars for W^Q, W^K, and W^V are static weights, their execution time still grows by n, as we need to compute the projection of the Q', K', and V' for each segment.) The number of required rows increases for K while the number of required columns increases for V as n increases. Since the rows can be treated independently of each other, $\lceil N/64 \rceil$ 64×64 arrays can be used to compute the MVM for K in parallel. Increasing the number of columns in V requires adding partial sums from the XBars, incurring additional latency. This additional latency, however, is still not significant at $n = 2^{12}$. The complexity of iMCAT, hence, is $O(n)$. Thus, iMCAT achieves a speedup close to $n/256$ compared to the GPU. This is seen by the rising speedup of the base implementation shown in Fig. 5(a). For much longer sequence lengths, LSH can reduce the number of columns required for V, thus, reducing this additional latency.

Because the total computations for both GPU and the base XBar implementation follow the complexity of $O(n^2)$, the increase in energy consumption remains the same, until $n > 2^{11}$, where the energy for the memory transfer overhead significantly starts to dominate for the GPU.

B. Improvement due to in-Memory Parallelism

For bidirectional transformers, the latency can be improved by introducing more parallelism via duplicating the XBars in each head by p (Sec. III-B3). This results in $p\times$ additional improvement in the execution time. By using XBars and parallelization ($p = n$), we can reduce the time complexity to $O(1)$ showing a latency improvement in Fig. 5(a).

Memory and power, however, are bounded. As an example, we set the memory size of each attention head to 15MB, shown in Fig. 5 as Parallel BM (Bounded Memory). Once the memory limit of iMCAT is reached, the execution time returns back to $O(n/p)$, where p is the number of attention-level parallelisms (achieved through duplication). At $n = 64$, this speedup equates to 673× speedup (in which $\approx 10\times$ speedup is from computing in-memory in XBars, and around $\approx 64\times$ speedup from parallelization). At $n = 4096$, a speedup of 5 orders of magnitude is achieved (10× from the base implementation, 100× from duplicating the XBars, and 100× from attention duplication).

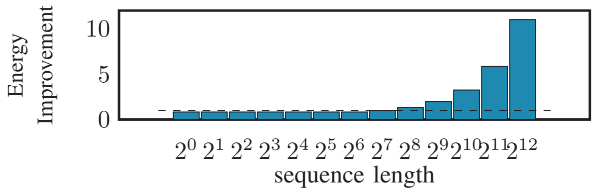

Fig. 6: Energy Improvement of using LSH+SDP MHA

Duplicating XBars, however, increases the number of writes and results in lower energy improvement. In the parallel (Fig. 5) scenario, increasing the attention level parallelism with respect to sequence length n also requires $n\times$ more writes (because $p = n$, resulting in a significant degradation in energy improvement. In the parallel BM (Fig. 5) scenario, there are $p\times$ available attention-level parallelism. If $n < p$, only $n\times$ more writes are required. At $n \geq p$, there are $p\times$ more writes.

The memory requirement also increases by a factor of n. However, when computing for the energy-delay product (EDP), more parallelism shows better EDP because the increase in energy is linear while the increase in delay is exponential. Having more parallelism results in a higher speedup but lower energy gains. More parallelism, however, results in a higher EDP. Hence, more parallelism is recommended within memory, energy, and other constraints.

C. Locality Sensitive Hashing + Dot-Product Attention

The accuracy of using LSH followed by SDP (LSH+SDP), is dependent on the signature length. We have found that we can achieve comparable accuracies with SDP when using a signature length of 1024 bits with a k-nearest neighbor of k=16 on the General Language Understanding Evaluation (GLUE) dataset. A pre-trained BERT [2] is used for the evaluation. The GLUE dataset is a common benchmark used in transformers

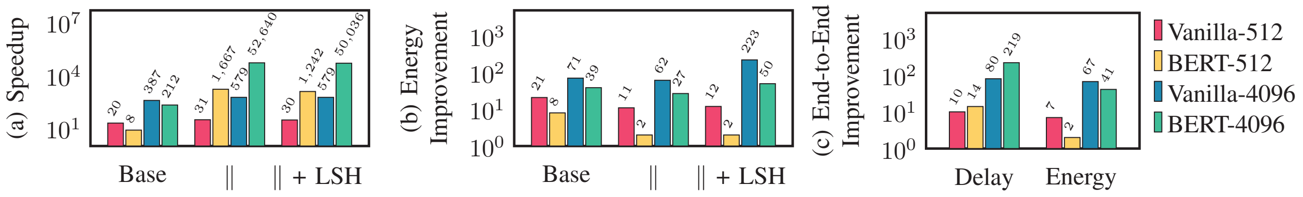

Fig. 7: (a) Speedup and (b) energy improvement of MHA and Feedforward layers for the Xbar only (base), duplicated Xbar with bounded memory (‖), duplicated Xbar-CAM with bounded memory using LSH+SDP MHA (‖ + LSH) implementation and (c) end-to-end speedup and energy improvement of vanilla and BERT transformers with sequence length 512 and 4096.

[2], [3]. This is also similar to using 64 buckets of 4 hashes each or $(64 \cdot 2^4)$ 1024 bits for the angular LSH used in the Reformer network.

Since we are only appending the LSH in a pre-trained network, an 11% (1.11×) increase in delay is incurred because of the CAM search. LSH reduces the number of dot-products required as the sequence length increases. The latency overhead still outweighs the latency improvement due to fewer attention computations to be performed at a sequence length of $n = 4096$. However, since CAM searches require fewer numbers of bits and are more energy-efficient than searching via dot-product on XBars, this results in an exponential improvement in energy (Fig. 6) as the sequence length increases. The bars below the dashed line (speedup=1) have worse execution time or a slowdown. Using LSH is only advisable for longer sequence lengths (greater than 2^8 or 256).

D. End-to-End Results

The feedforward and MHA layers are accelerated in-memory while the layer normalization is sent back to the CPU/GPU for processing. Fig. 7 shows the delay and energy improvement of the feedforward and MHA with parallelism and LSH enhancements. The base implementation (without attention level parallelism) achieves a speedup of 20× and 8× for the vanilla transformer and BERT, respectively, when $n = 512$. This increases to 387× and 212× at $n = 4096$. For the vanilla transformer, only the encoder layers can be parallelized, as opposed to bidirectional transformers such as BERT, where all layers can be parallelized. The speedup gained from attention parallelization is hence smaller for vanilla transformers than bidirectional transformers as the sequence length increases. A speedup of 2 orders of magnitude (≈248×) is achieved by parallelizing attention. LSH adds additional latency but the latency is not significant, as shown in Fig. 7. By including the layer normalization, iMCAT can achieve a 200× delay and 41× energy improvement for BERT at $n = 4096$, as shown in Fig. 7.

V. Conclusion

We have shown iMCAT, an in-Memory XBar-CAM implementation of transformer networks, minimizes the memory transfer overhead. Attention-level parallelization is used to improve the latency and LSH is used to achieve better energy consumption. We were able to achieve a 20× latency and energy improvement at a sequence length of $n = 512$ for the multi-head attention portions of the transformer network by computing-in-memory using XBars, and reusing attention feature embeddings. By parallelizing bidirectional multi-head attention, we can increase our latency significantly but this also increases the energy consumption. We achieved a 200× end-to-end speedup and 41× energy improvement compared to the GPU implementation at a sequence length of $n = 4096$.

Acknowledgment

This work was supported in part by ASCENT, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA.

References

[1] A. Vaswani, et al. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.
[2] J. Devlin, et al. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.
[3] Z. Lan, et al. ALBERT: A lite BERT for self-supervised learning of language representations. In *ICLR*, 2020.
[4] M. Shoeybi, et al. Megatron-lm: Training multi-billion parameter language models using model parallelism. *CoRR*, abs/1909.08053, 2019.
[5] T. B. Brown, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.
[6] Z. Yang, et al. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in Neural Information Processing Systems, NeurIPS*, 2019.
[7] N. Kitaev, et al. Reformer: The efficient transformer. In *8th International Conference on Learning Representations*, 2020.
[8] O. Zafrir, et al. Q8bert: Quantized 8bit bert. *arXiv*, 2019.
[9] Z. Li, et al. Train large, then compress: Rethinking model size for efficient training and inference of transformers. *arXiv*, 2020.
[10] A. F. Laguna, et al. Design of hardware-friendly memory enhanced neural networks. In J. Teich et al., editors, *Design, Automation & Test in Europe Conference & Exhibition*, pages 1583–1586. IEEE, 2019.
[11] K. Ni, et al. Ferroelectric ternary content-addressable memory for one-shot learning. *Nature Electronics*, 2(11):521–529, Nov 2019.
[12] N. Challapalle, et al. Farm: A flexible accelerator for recurrent and memory augmented neural networks. *Journal of Signal Processing Systems*, Jun 2020.
[13] A. Ranjan, et al. X-mann: A crossbar based architecture for memory augmented neural networks. pages 1–6, 06 2019.
[14] D. Reis, et al. A fast and energy efficient computing-in-memory architecture for few-shot learning applications. In *2020 Design, Automation & Test in Europe Conference & Exhibition*. IEEE, 2020.
[15] A. F. Laguna, et al. Ferroelectric FET based in-memory computing for few-shot learning. In H. Homayoun, et al., editors, *Proceedings of the 2019 on Great Lakes Symposium on VLSI*. ACM, 2019.
[16] T. Gokmen et al. Acceleration of deep neural network training with resistive cross-point devices: Design considerations. *Frontiers in neuroscience*, 10:333, 2016.
[17] I. Kouretas et al. Hardware implementation of a softmax-like function for deep learning. *Technologies*, 8(3):46, 2020.
[18] G. Prato, et al. Fully quantized transformer for improved translation. *CoRR*, abs/1910.10485, 2019.