MAT: Processing In-Memory Acceleration for Long-Sequence Attention

Minxuan Zhou
University of California, San Diego
miz087@ucsd.edu

Yunhui Guo
University of California, San Diego
yug185@ucsd.edu

Weihong Xu
University of California, San Diego
wexu@ucsd.edu

Bin Li
University of Wisconsin-Madison
bli346@wisc.edu

Kevin W. Eliceiri
University of Wisconsin-Madison
eliceiri@wisc.edu

Tajana Rosing
University of California, San Diego
tajana@ucsd.edu

2021 58th ACM/IEEE Design Automation Conference (DAC) | 978-1-6654-3274-0/21/$31.00 ©2021 IEEE | DOI: 10.1109/DAC18074.2021.9586212

***Abstract*—Attention-based machine learning is used to model long-term dependencies in sequential data. Processing these models on long sequences can be prohibitively costly because of the large memory consumption. In this work, we propose MAT, a processing in-memory (PIM) framework, to accelerate long-sequence attention models. MAT adopts a memory-efficient processing flow for attention models to process sub-sequences in a pipeline with much smaller memory footprint. MAT utilizes a reuse-driven data layout and an optimal sample scheduling to optimize the performance of PIM attention. We evaluate the efficiency of MAT on two emerging long-sequence tasks including natural language processing and medical image processing. Our experiments show that MAT is 2.7× faster and 3.4× more energy efficient than the state-of-the-art PIM acceleration. As compared to TPU and GPU, MAT is 5.1× and 16.4× faster while consuming 27.5× and 41.0× less energy.**

I. Introduction

Self-attention has emerged as a very powerful tool to model long-term dependencies in sequential data, such as language modeling [19]. The recently proposed transformer architecture [19] has been considered as a better substitute for recurrent neural networks (RNN) [16] and demonstrated the state-of-the-art performance for natural language processing (NLP) applications. Self-attention has also been applied to image analysis as the form of a non-local layer which is beneficial for its ability to capture relations of image features that span the whole input image [20]. Recently, the transformer architecture has also been applied to computer vision tasks for image classification and object detection [2], [3] where the input images are cropped into patches and assumed dependencies like sequences. For language modeling, self-attention differs from RNN in that all the words in the sentence are examined at once and long-term dependencies are captured regardless of the distance between the words. Similarly, for vision tasks, in contrast to convolution which involves the operation of a convolution kernel on each local position, self-attention operates on the whole image and models the feature relations in all the positions across the image. Similar attention mechanisms have also been used in multiple instance learning (MIL) problem for weakly supervised object detection [9], [14].

One major difference between self-attention and convolution as well as RNN is that self-attention models the interactions of all words, and requires the whole sequence to be observed at once. Typically, an embedding phase is utilized to generate embedding vector for each word in the sequence and then all the embedding vectors are used as input for self-attention model for computing pair-wise score matrix. A significant challenge to accelerate attention models is the prohibitive memory cost due to the need to store the score matrix in the case of long sequences. Though generally being more memory-efficient as compared to RNNs, the memory footprint increases quadratically with the sequence length in self-attention models. As compared to other popular models like convolutional neural networks (CNNs), the attention models are more memory-bound because of the low compute density in terms of the large amount of weights. Therefore, conventional accelerators optimized for convolution layers and fully-connected layers may not perform well for attention models. Our evaluation on a memory-efficient PyTorch implementation on GPU [13] shows that the GPU memory cannot fit an attention layer with a sequence length of 60K. Furthermore, the memory-efficient implementation only processes a portion of attention at a time to reduce the memory footprint. Its performance is much slower than the original implementation because of frequent data loading.

PIM is a good way to accelerate attention models, since they are memory-bound but with a large amount of parallel operations. Even though there are many PIM-based accelerators for machine learning models, most of them are designed for CNNs [4], [10], [15], [18] which have good data reuse rate to schedule parallel operation with moderate loading cost. However, for the attention model, the data flow used by previous PIM accelerators may be prohibitively costly. In this paper, we propose, MAT, a new processing paradigm based on digital PIM (DPIM) technology to accelerate long-sequence attention models. MAT adopts a iterative tiled processing scheme based on a modified computation order in the attention layer. The iterative tiled processing scheme enables us to adjust different amount of computations to achieve better memory utilization. Furthermore, we propose two optimization techniques, reuse-driven data layout and scheduling optimization, to optimize the

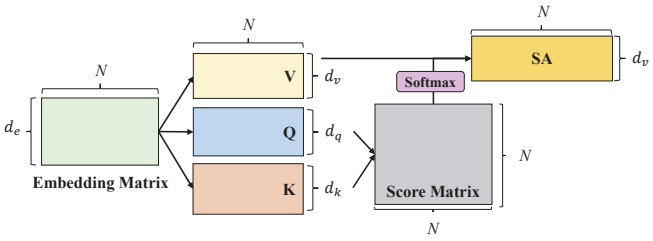

Fig. 1. Operations in an attention layer.

performance of DPIM-based attention.

We demonstrate the efficacy of MAT on two common applications of self-attention models — sentence-to-sentence translation and weakly supervised tumor detection in whole slide images [7]. We compare the performance of MAT with various state-of-the-art systems including TPU, GPU, and existing PIM accelerators. Our experiments show that MAT is 2.7× faster and 3.4× more energy efficient than the state-of-the-art PIM acceleration. As compared to TPU and GPU, MAT is 5.1× and 16.4× faster while consuming 27.5× and 41.0× less energy.

II. Background and Motivation

A. Long-sequence Attention Model

Attention-based models typically consist of an encoder and a decoder [19]. Both the encoder and the decoder are constructed by stacking multiple self-attention layers. The key operation in a self-attention layer is the scaled dot-product attention as shown in Fig. 1. For each piece of data (word) in the input sequence, we generate an embedding vector of dimension d_e. For a sequence of length N, all the embedding vectors can be regarded as an embedding matrix of dimension $d_e \times N$. For each word we generate a query vector $\boldsymbol{Q}$ of dimension d_q and a key vector $\boldsymbol{K}$ of dimension d_k, and a value vector $\boldsymbol{V}$ of dimension d_v by multiplying the embedding vector with different weight matrices. One typical value of d_k, d_q and d_v is 512. The $\boldsymbol{Q}$ vectors, $\boldsymbol{K}$ vectors and $\boldsymbol{V}$ vectors of all the words in the input sequence can be stacked together as matrices $\boldsymbol{Q}$, $\boldsymbol{K}$ and $\boldsymbol{V}$. In the scaled dot-product attention, the dependencies between words can be captured via the following equation:

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

where Softmax$(\cdot)$ denotes the Softmax function. The $\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_k}}$ is defined as the score matrix $\boldsymbol{S}$ that measures how much focus to put on various parts of the input sequence. We use $\boldsymbol{S}\boldsymbol{A}$ to denote the output of the attention layer.

To achieve higher accuracy and performance, the training of large attention models is conducted on long sequences [9], [13], [14]. The long sequences pose a severe challenge for accelerating attention models due to the prohibitive memory cost. The attention layer requires $O(N^2)$ memory space to fully parallelize the computation and store intermediate results, where N is the sequence length. In this case, the memory footprint increases quadratically with the sequence length. Concretely, the bottleneck in self-attention is the computation of the score matrix $\boldsymbol{S} = \frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_k}}$ in Equation 1, where $\boldsymbol{S}$ has a size of $N \times N$.

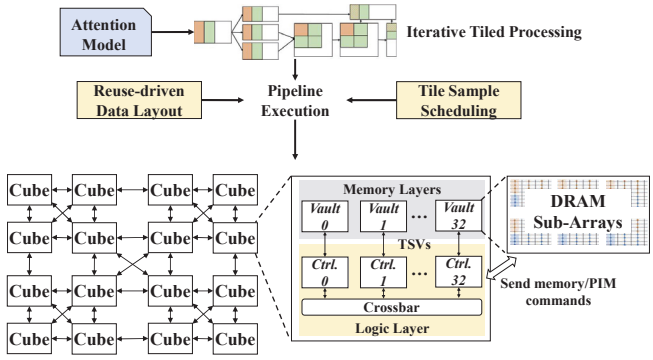

Fig. 2. Overview of MAT on a DPIM-enabled HMC System.

For typical natural language processing tasks, the score matrix consumes 16GB of memory when the sequence length N is $64K$ and using 32-bit floating-point precision. When further scaling the sequence length, the memory footprint is beyond the capacity of most commodity GPUs.

B. Digital Processing in-Memory Acceleration

Digital processing in-memory (DPIM) technology is used to accelerate memory-bound applications. Logic operations (e.g., NOT, AND, OR, NOR, NAND, etc.) between memory cells in two different memory rows are basic PIM operations. Several previous works have implemented these DPIM operations in various memory technologies, including DRAM [6], [15], SRAM [4], and ReRAM [8]. For example, ComputeDRAM [6] shows how to enable DPIM operations in the off-the-shelf DRAMs by charge sharing when executing a sequence of ACTIVATE and PRECHARGE commands without idle cycles. FELIX proposes several single-cycle logical operations in resistive memory crossbar without changing sense amplifiers [8]. Such single-bit logical operations can be used to implement arbitrary computations in multiple computation steps (bit-serial) including multi-bit addition and multiplication. A single DPIM computation can simultaneously affect all values in a memory rows (row-parallel). These bit-serial row-parallel operations can achieve massive parallelism because all memory blocks can execute DPIM operations in parallel.

There have been several existing DPIM-based accelerators to accelerate machine learning models [4], [10], [15]. Although they can be directly applied to compute attention models, most of them are designed for Convolutional Neural Networks (CNNs), which have tremendous data reuse opportunities and are computation-intensive. However, attention models have a lower computational intensity compared to CNNs. Computing the attention requires a large amount of memory to store the weights and the inference of attention models is dominated by memory accesses instead of computation. The memory and computation imbalance between attention models and DPIM accelerators results in low hardware efficiency and utilization due to the frequent data loading.

III. MAT Design

In this work, we propose, MAT, a processing paradigm of DPIM acceleration based on emerging hybrid memory

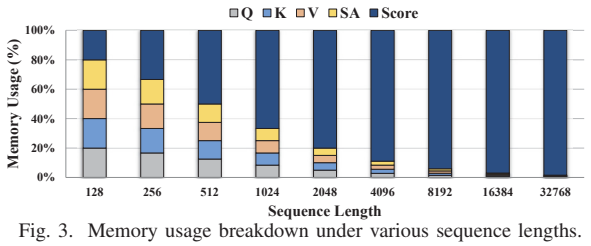

Fig. 3. Memory usage breakdown under various sequence lengths.

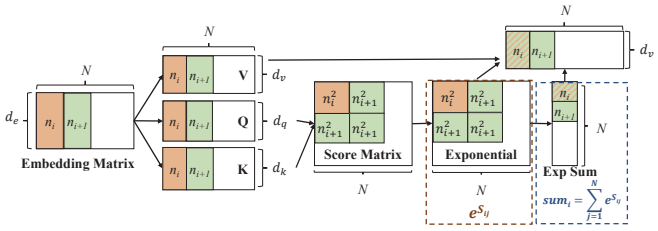

Fig. 4. Iterative tiled processing flow with late softmax update.

cube (HMC) architecture for attention models to address the aforementioned challenges.

A. System Overview

Fig. 2 shows the underlying architecture used by MAT, which is a DPIM-enabled DRAM system based on HMC architecture [6], [11]. An HMC system consists of multiple memory cubes that are connected by high-speed serial links in an interconnect network. Each memory cube consists of 3D-stacked memory layers with one logic layer. The logic layer has multiple controllers, each of which control a vertical partition of memory (vault) containing many memory sub-arrays. Memory controllers transfer data from memory layers with through silicon vias (TSVs). We adopt the DRAM-based DPIM technology validated in off-the-shelf DRAM [6] which only requires moderate modifications in the memory controller to send DPIM commands to the sub-arrays. Detailed architectural parameters used for our evaluation are discussed in Section IV.

MAT adopts an iterative tiled processing flow which processes different parts of the computation through all layers to improve the memory and hardware utilization. The iterative tiled processing is scheduled in a pipelined execution which adopts two optimization techniques, reuse-driven data layout and optimal sample scheduling, to further improve the efficiency of processing flows.

B. Iterative Tiled Processing Flow

It is costly to load and process the entire input embeddings since the intermediate matrices would consume huge amount of memory space. We plot the memory usage breakdown for five types of matrices for attention models in Fig. 3. It shows that the score matrix $\boldsymbol{S}$ accounts for the majority of total memory usage as the sequence length is greater than 1024. Hence, reducing the memory requirement of score matrix is a key design factor.

To avoid buffering the whole score matrix, we propose an iterative tiled processing flow through modifying the computation order and dataflow of an attention layer. Fig. 4 shows details of the iterative tiled processing for attention models. The basic idea is to divide data into small portions, therefore saving the space required to store the memory-consuming score matrix. Specifically, the PIM system pre-allocates the memory resources for the attention block, including $\boldsymbol{Q}$, $\boldsymbol{K}$, $\boldsymbol{V}$, the score matrix $\boldsymbol{S}$, the exponential matrix, the exponential sum, and the self-attention output $\boldsymbol{SA}$. At the i-th iteration, the input embeddings with sequence length N are partitioned into tiles with a shorter length n_i. Then the length-n_i tile of input embeddings is fed into the PIM accelerator and used to generate its corresponding tiles of $\boldsymbol{V}$, $\boldsymbol{Q}$, and $\boldsymbol{K}$ matrices. The tiled score matrix $\boldsymbol{S}$ is computed using all the tiles of $\boldsymbol{Q}$ and $\boldsymbol{K}$. $2i-1$ new tiles of the score matrix are generated at the i-th iteration. Then the point-wise exponential of the score matrix $e^{S_{i,j}}$ is computed and the tiled point-wise exponential matrix is multiplied with the tiled $\boldsymbol{V}$ matrix, yielding the partial sum of the self-attention matrix. Finally, the aggregation of self-attention matrix is divided by the exponential sum to normalize the values into probability domain and generate the final self-attention results. The details are introduced in Section III-C. Each input tile is processed sequentially in a pipelined manner to improve the memory utilization, which is explained in Section III-D.

C. Late Softmax Update

The naive attention dataflow does not support the iterative tiled processing scheme because there is an all-to-one data dependence between the outputs of score matrix and Softmax. Specifically, the results of Softmax in Eq. (1) can only be calculated after obtaining all the results of the score matrix $\boldsymbol{S}$ since the denominator of Softmax is the summation of all elements in a row of $\boldsymbol{S}$. In order to relax this data dependency, we propose a "late softmax update" mechanism which postpones the denominator calculation of Softmax. The element of $\boldsymbol{SA}$ matrix in the i-th row and j-th column, $\boldsymbol{SA}_{i,j}$, is calculated as:

$$\boldsymbol{SA}_{i,j} = \sum_{k=1}^{N} \frac{e^{\boldsymbol{S}_{i,k}}}{\sum_{l=1}^{N} e^{\boldsymbol{S}_{i,l}}} V_{k,j} = \frac{1}{\sum_{l=1}^{N} e^{\boldsymbol{S}_{i,l}}} \sum_{k=1}^{N} e^{\boldsymbol{S}_{i,k}} V_{k,j}, \tag{2}$$

where the computation order of the denominator is put on the outermost. The exponential value of $\boldsymbol{S}_{i,k}$ is first multiplied with the corresponding element of $\boldsymbol{V}$ matrix without knowing the summation of denominator.

To support the late softmax update, we change the original Softmax layer to an exponential layer and an exponential sum layer as shown in Fig. 4. Specifically, the exponential layer calculates the exponential value of each element in the score matrix. The exponential computation is approximated as $e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} \ldots$ using the Taylor Series Expansion, allowing the PIM architecture to compute exponential values in parallel with acceptable accuracy. The exponential sum layer stores and updates partial sums of each normalization term across iterations. Then the accumulated sums are used to normalize the output $\boldsymbol{SA}$ matrix at the end of the last iteration.

D. Pipelined Operation Scheduling

In DPIM acceleration, we allocate exclusive memory resources for different layers in the attention models. During one iteration for a tile, only a portion of $\boldsymbol{V}$, $\boldsymbol{Q}$, and $\boldsymbol{K}$ matrices

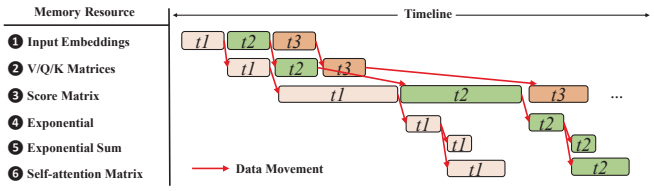

Fig. 5. The pipeline of tiled processing flow.

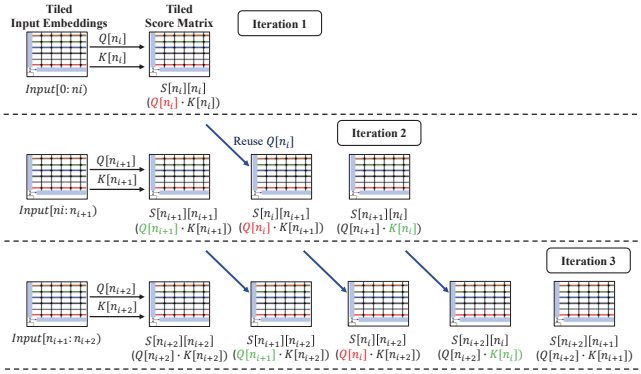

Fig. 6. An example of data reuse between three iterations.

are activated. In this case, the iterative tiled processing enables a pipelined execution to improve the memory utilization.

1) Pipeline Execution: Fig. 5 shows a timeline for pipelining computations of different tiles across iterations. In each iteration, the DPIM system sequentially processes layers based on the data dependency. Such dependency requires data movement between the memory resources allocated to consecutive layers. For example, we need to send the output of $\boldsymbol{Q}$, and $\boldsymbol{K}$ vectors to Score matrix. When a module (e.g. compute $\boldsymbol{V}$ matrix) finishes all computations for the current tile, it can start processing the next tile after loading new inputs or weights.

When processing a new tile, each layer needs to load the data required for computations in the memory. Such data may be the input samples for feature extraction, the trained weights in fully-connected layers, and the output of previous layers. The input samples and the trained weights are stored in the separate memory area as the normal data in conventional memories. The output results from the previous layers are stored in the memory area allocated for those layers. The pipeline execution can effectively hide the overhead of data loading.

The required memory space is reduced after adopting the tiled processing dataflow. The original attention dataflow requires $\mathcal{O}(N^2)$ memory space to buffer and process the score matrix. In comparison, MAT only consumes $\mathcal{O}(n_i^2)$ space, where n_i denotes the tile size and is much smaller than N. The memory consumption is adjustable across iterations by selecting different tile sizes. The optimal scheduling of samples is determined by the optimization framework in Section III-D3.

2) Reuse-Driven Data Layout: After completing computations for a layer in one iteration, new data needs to be loaded into the memory for the next iteration. Such data includes weights for the new tile and outputs from the previous layer. Loading data may take a significant portion of time because of the large data size of attention models. Furthermore, the

Algorithm 1: Dynamic programming based optimization framework for sample scheduling.

```
Input: n_samples, pim_system
f = array(n_samples);
step = array(n_samples);
f[0] = 0;
for i ← 1 to n_samples do
    for j ← 1 to i do
        if can_process(j, i, pim_system) then
            if f[i − j] + process(j, i, pim_system) < f[i] then
                f[i] = f[i − 1] + process(j, i, pim_system);
                step[i] = j;
```

data size required for each iteration increases over time and the repeated data loading degrades the efficiency and becomes a performance bottleneck due to the waste of memory bandwidth. To exploit the reuse opportunities, we propose a reuse-driven data layout to reduce the overhead of data loading. Fig. 6 shows the proposed reuse-driven data layout for feature extraction layer and self-attention layer over three consecutive iterations. In the first iteration, DPIM system allocates two memory segments for feature extraction and self-attention. The output of feature extraction ($\boldsymbol{Q}$ and $\boldsymbol{K}$) is loaded to the self-attention memory. In the second iteration, the feature extraction memory generates $\boldsymbol{Q}$ and $\boldsymbol{K}$ for new tile (n_{i+1}). The self-attention memory, instead of loading all data, can reuse the memory segment in the first iteration without loading the existing data ($Q[n_i]$). Similarly, in the third iteration, we can reuse data in three memory segments of the second iteration. Therefore, there are only two memory segments that have to load completely new data in each iteration, significantly reducing the overhead of data loading.

3) Optimal Sample Scheduling: There remains one key design in the MAT execution model - the input sample scheduling, which selects different numbers of samples to be processed in each iteration. We introduce optimization techniques for both aspects that can significantly improve the efficiency of DPIM acceleration. As mentioned in Section III, we need to schedule the samples processed in each iteration. If we use a fixed number of samples, PIM system would either waste most of its computing throughput or may not meet the hardware constraints. Therefore, we propose a optimization algorithm based on dynamic programming that can find the optimal sample scheduling within the hardware constraints. Algorithm 1 shows the steps of the proposed algorithm. The inputs of the algorithm are the number of total samples (sequence length) and the hardware configuration of PIM system. We define a state function f with $n\_samples$ element, where $f[i]$ represents the lowest cost of processing the first i samples. For each $f[i]$, the lowest cost is calculated by $min(f[i-j]+process(j,i,pim\_system)$, where j is a number from 1 to i and $process(j,i,pim\_system)$ is the performance cost of processing j samples in one iteration with i total samples. The performance cost is estimated from a hardware configurable cycle-accurate simulation which is introduced in Section IV. We use a $step$ array to record the optimal step for each i to get the optimal schedule.

IV. Experiments

In this section, we introduce the methodology used for our evaluation and experimental results.

A. Experimental Setup

TABLE I
ARCHITECTURAL PARAMETERS

	Configuration
HMC	8GB/cube, 500MHz bus/command frequency 32 vaults, 16MB banks, 8KB row buffer Bandwidth: 512GB/s (internal), 320GB/s (external)
t_{CAS}-t_{RCD}-t_{RP}-t_{RAS} t_{RC}-t_{WR}-t_{WTR}-t_{RTP} t_{RRD}-t_{FAW}	7-7-7-17 24-8-4-7 5-20
PIM Operations (per bit)	Row copy: 18 cycles, SHIFT: 36 cycles AND/OR: 172 cycles, XOR: 444 cycles ADD: 1332 cycles

1) Simulation: We use Cacti [17]), and published papers [6] to characterize the timing and energy models for different PIM operations. Based on these models, we implement an in-house simulator to model the performance of PIM acceleration for running the workloads in PyTorch. In this work, we use DRAM [6] as the underlying technology and the hybrid memory cube (HMC) [11] as the basic architecture. Table I shows the parameters of PIM and HMC technology used in this work, which are calculated based on previous work [1], [6], and published data [11]. We use the largest HMC 2.1 configuration, which has eight 8Gb layers per cube with 32 vertically partitioned memory vaults. Each vault has a dedicated memory controller that can schedule PIM operations in the memory [6]. The baseline system has 16 cubes (128GB in total) linked with eight 40 GB/s high-speed serial links per cube in a dragon-fly interconnect [1].

2) Datasets: We use enwik8-64K [19] for natural language processing and Camelyon16 [5] for whole slide image classification to demonstrate the effectiveness of the proposed framework.

For natural language processing, the enwik8-64K dataset is separated into different numbers of subsequences (from 5K to 95K tokens) which allows us to evaluate the performance of the framework in the case of different sequence lengths. We use a 3-layer attention model to make it tractable to simulate, which has high memory usage and performs full O(N^2) attention.

For whole slide image classification, the dataset consists of 400 whole slide images with an average size of $30,000 \times 30,000$ pixels per image at the magnification of $20\times$. We use this dataset for weakly supervised tumor detection [14]. To simulate long sequences, we cut the image into small patches and reshape the patches into 224×224. These operations allow us to have 5K to 95K patches per image. The patches from an image will be projected into 512×1 feature vectors using a pretrained ResNet18 and fed into self-attention model for aggregation and inference. In all the experiments we use $d_v = d_k = d_q = 512$.

3) Baseline Systems: We use PyTorch to implement the inference of different models running on GPU. The GPU platform used in experiments is NVIDIA TITAN RTX. We also implement a roofline model for TPU 1.0 based on its published design description [12]. For the baseline PIM acceleration, we use the processing flow used by previous DPIM CNN accelerators [4], [10], [15]. We should note that we only use the processing flow of previous work, while adopting them to the same DRAM architecture for a fair comparison. The PIM baseline sequentially processes attention layers. When processing layers with a memory footprint larger than system capacity (e.g., attention), the PIM baseline iteratively processes small portions to fit computation in the memory. All systems run with 8-bit precision.

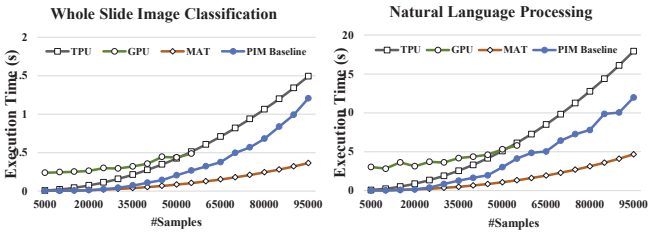

Fig. 7. The execution time of two models on different platforms.

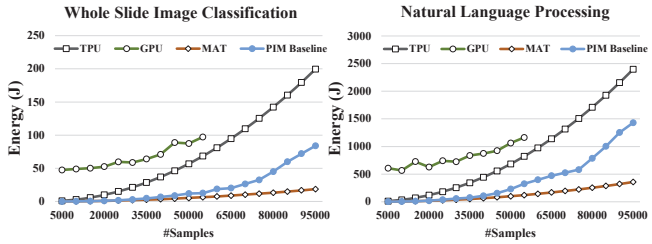

Fig. 8. The energy consumption of two models on different platforms.

B. Results

1) Comparison with existing platforms: Fig. 7 shows the execution time of two long-sequence attention models on TPU, GPU, and PIM baseline and MAT. We should note that GPU baseline fails to run the sequence longer than 60,000 because of the out-of-memory errors. The results show that MAT is $5.2\times$ and $4.9\times$ faster than TPU for the whole slide image classification and natural language processing respectively. As compared to GPU, MAT provides $16.6\times$ and $16.1\times$ speedup on two models. The results show that MAT can significantly improve the performance of attention models and it has efficient scalability as a function of the sequence length.

MAT is $2.4\times$ and $2.9\times$ faster than the PIM baseline. The performance improvements of MAT becomes larger when increasing the sequence length. This is because the PIM baseline requires more data loading to process the large attention matrix with longer sequences. Such data loading becomes the performance bottleneck since the PIM baseline uses all available memory for attention. In this case, the baseline PIM stalls during the data loading, which significantly lowers the memory utilization for computations.

Fig. 8 shows the energy consumption of three systems. As compared to TPU (GPU), MAT consumes $33.8\times$ ($49.9\times$) and $21.2\times$ ($31.8\times$) less energy on two models. The energy benefits of MAT over the conventional systems come from the faster execution and the removal of off-memory data transfer. As compared to PIM baseline, MAT consumes $3.2\times$ and $3.6\times$ less energy because of the faster execution.

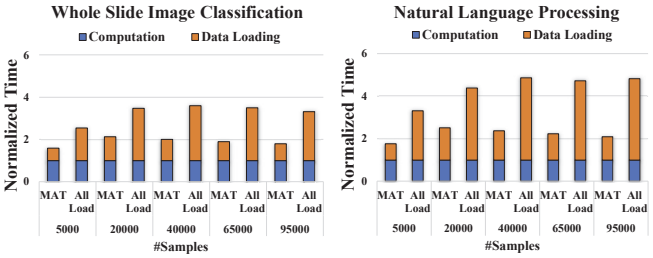

Fig. 9. The effects of reuse-driven data layout.

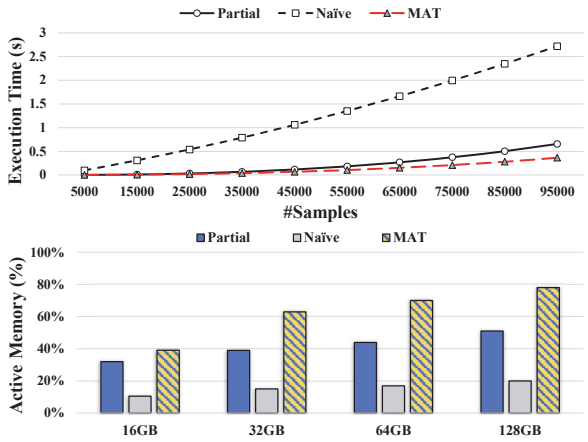

Fig. 10. The performance and memory usage of different operation scheduling methods.The utilization experiments are done on various system size for 60,000 long sequences. Both results are on the whole slide image classification model.

2) Effect of Reuse-Driven Data Layout: Fig. 9 shows the effects of reuse-driven data layout. We compare MAT to a baseline system that loads all data for computations without any data reuse ("All Load" in Fig. 9). The experiment shows that reuse-driven layout can reduce the latency of data loading by 2.7× and 3.1× respectively, resulting in 1.82× and 1.94× speedups for the whole slide image classification and the natural language processing models respectively. Between two models, natural language processing requires longer time for data loading because of the large vector length for $\boldsymbol{V}$, $\boldsymbol{Q}$, and $\boldsymbol{K}$. The results show the proposed technique works better on larger models.

3) Sample Scheduling Methods: Fig. 10 shows the comparison between different sample scheduling methods on the whole slide image classification model. Specifically, the partial scheme processes the original softmax layer, and only pipelines the layers up to the softmax layer. "Partial" scheme then uses the whole attention matrix after the softmax to compute the output. "Naive" schemes set a fixed number of samples processed in each iteration. We select the maximum values that do not violate the hardware constraints. The results show that the scheduling of MAT is 10.6× and 18.2× faster than the "Partial" and "Naive" schemes respectively.

We also investigate the memory utilization of different scheduling, where the memory utilization indicates what is the average portion of memory are active on computation during the execution. Fig. 10 shows average percentage of active memory of three methods on different sized systems. To simplify the plots, we only show the result for 60,000-long sequences computation. The result shows that MAT improves the memory utilization of "Partial" and "Naive" scheduling by 1.2× and 3.7× on a 16GB. When increasing the system size by 4× (128GB), MAT has 1.6× and 4.1× better utilization than two other methods, and increases the memory utilization from 39% to 70%. The better utilization on larger system results from the more memory resources for the optimization to find more efficient scheduling.

V. Conclusion

In this work, we propose a new processing paradigm using PIM-enbaled HMC architecture for long-sequence attention models. The proposed method adopts a iterative tiled processing method to provide a memory-efficient processing to compute different attention models. We propose several optimization techniques to reduce the data loading overhead and find the optimal operation scheduling. Our experiments show that MAT is 2.7× faster and 3.4× more energy efficient than previous PIM acceleration. As compared to TPU and GPU, MAT is 5.1× and 16.4× faster while consuming 27.5× and 41.0× less energy.

Acknowledgment

This work was partially supported by CRISP, one of six centers in JUMP, an SRC program sponsored by DARPA, and also NSF grants #1730158 and #1527034.

References

[1] J. Ahn et al. A scalable processing-in-memory accelerator for parallel graph processing. In *ISCA'15*. ACM/IEEE.
[2] Nicolas Carion et al. End-to-end object detection with transformers. *arXiv preprint arXiv:2005.12872*, 2020.
[3] Alexey Dosovitskiy et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.
[4] Charles Eckert et al. Neural cache: Bit-serial in-cache acceleration of deep neural networks. In *ISCA'18*. ACM/IEEE.
[5] Babak Ehteshami Bejnordi et al. Diagnostic Assessment of Deep Learning Algorithms for Detection of Lymph Node Metastases in Women With Breast Cancer. *JAMA*, 2017.
[6] Fei Gao et al. Computedram: In-memory compute using off-the-shelf drams. In *MICRO'19*. ACM/IEEE.
[7] Farzad Ghaznavi et al. Digital imaging in pathology: Whole-slide imaging and beyond. *Annual Review of Pathology: Mechanisms of Disease*, 2013.
[8] Saransh Gupta et al. Felix: Fast and energy-efficient logic in memory. In *ICCAD'18)*, pages 1–7. IEEE, 2018.
[9] Maximilian Ilse et al. Attention-based deep multiple instance learning. *ICML'18*, 2018.
[10] Mohsen Imani et al. Floatpim: In-memory acceleration of deep neural network training with high precision. In *ISCA'19*. ACM/IEEE.
[11] Joe Jeddeloh and Brent Keeth. Hybrid memory cube new dram architecture increases density and performance. In *VLSIT'12*. IEEE.
[12] Norman P Jouppi et al. In-datacenter performance analysis of a tensor processing unit. In *ISCA'17*, pages 1–12, 2017.
[13] Nikita Kitaev et al. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020.
[14] Bin Li et al. Dual-stream multiple instance learning network for whole slide image classification with self-supervised contrastive learning. *arXiv preprint arXiv:2011.08939*, 2020.
[15] Shuangchen Li et al. Drisa: A dram-based reconfigurable in-situ accelerator. In *MICRO'17*. ACM/IEEE.
[16] Zachary C Lipton et al. A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019*, 2015.
[17] Naveen Muralimanohar et al. Optimizing nuca organizations and wiring alternatives for large caches with cacti 6.0. In *MICRO'07*. ACM/IEEE.
[18] Ali Shafiee et al. Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. *ACM SIGARCH Computer Architecture News*, 44(3):14–26, 2016.
[19] Ashish Vaswani et al. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.
[20] Xiaolong Wang et al. Non-local neural networks. In *CVPR'18*, 2018.