Accelerating Sparse Attention with a Reconfigurable Non-volatile Processing-In-Memory Architecture

Qilin Zheng, Shiyu Li, Yitu Wang, Ziru Li, Yiran Chen and Hai (Helen) Li
{qilin.zheng, shiyu.li, yitu.wang, ziru.li, yiran.chen, hai.li}@duke.edu
Dept. of ECE, Duke University

***Abstract*—Attention-based neural networks have shown superior performance in a wide range of tasks. Non-volatile processing-in-memory (NVPIM) architecture shows its great potential to accelerate the dense attention model. However, the unique unstructured and dynamic sparsity pattern in the sparse attention model challenges the mapping efficiency of the NVPIM architecture, as the conventional NVPIM architecture uses a vector-matrix-multiplication primitives. In this paper, we propose a NVPIM architecture to accelerate a dynamic and unstructured sparse computation in the sparse attention. We aim to improve the mapping efficiency for both SDDMM and SpMM by introducing two vector-based primitives with a reconfigurable NVPIM bank. Further, based on our reconfigurable NVPIM bank, we further propose a hybrid stationary data flow to hide the latency. Our evaluation result shows that, over previous NVPIM accelerators, our design could deliver up to $12.36\times$ performance improvement and $3.4\times$ energy efficiency improvement on a range of vision and language tasks.**

I. Introduction

With the development of machine learning, attention-based neural networks are being viewed as the next generation solution for computer vision (CV) [1] and natural language processing (NLP) [2]. The input vectors are first projected into query (Q), key (K) and value (V) matrices through linear transformation. Then, self-attention is operated in two consecutive stages. In the first stage (attention computation), the attention map is generated by normalizing the product of Q and K with the softmax function. In the second stage (attention computation), the normalized attention map is multiplied by the V matrix to generate the output. This computational complexity hinders the deployment of attention-based model to resource-constrained devices.

Various solutions has been proposed to address the computational cost either from the algorithm or hardware perspectives. Sparse attention, where redundant attention scores are identified and eliminated, is proposed to address the computational cost from the algorithm perspective. The sparse attention could reach over 90% sparsity with a irregular and dynamic generated sparse map. From the hardware perspective, non-volatile processing-in-memory (NVPIM) architectures have been proved to be a promising solution to address the computation need of attention-based models [3]. The NVPIM architecture can perform the vector-matrix multiplication (VMM) operations efficiently via analog computing on the memory array. Furthermore, it can substantially reduce the overhead of data movement, as the computation happens *in-situ* inside the memory array.

However, it is not easy to combine the optimal solution from both worlds. Although there are non-PIM accelerators to support sparse attention [4], [5], the cost of data movement could potentially be the bottleneck and offsets the benefit of sparsity. Meanwhile, the irregular and dynamic nature of sparse attention maps makes it difficult to map sparse attention to existing NVPIM-based designs. Some existing works, such as SRE [6], applied a structured pruning method at the bit-level for both activations and weights. However, their method fails to achieve substantial speed up due to a dynamic and unstructured sparsity pattern in the attention-based model. As shown in Fig. 1, directly mapping a *transformer* model with dynamic and unstructured

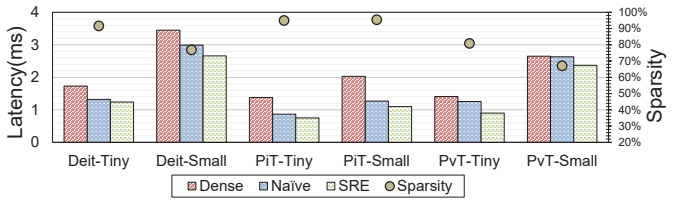

Fig. 1. Limited latency reduction of the previous NVPIM architecture with sparse attention models.

sparse patterns to the ReTransformer architecture will only induce about $1.59\times$ ($1.0\times$) speedup under 95% (67%) sparsity level. Even equipping the baseline with the method proposed in SRE, the speedup only reaches about $1.84\times$ ($1.11\times$) under the same sparsity level, which is far from the theoretical savings. As the native processing granularity of NVPIM is matrix, the utilization rate of the NVPIM banks will drop significantly under the sparse scenario.

To fill this gap and release the potential of the NVPIM architecture to process dynamic and unstructured sparse patterns, we propose a NVPIM architecture with a reconfigurable NVPIM bank to accelerate the sparse attention-based model. We use an architecture-circuit co-design method to improve the mapping efficiency of both SDDMM and SpMM. The basic idea is to provide two additional vector-based computation primitives to support the sparse computation pattern by reusing the interface circuits in the conventional NVPIM bank. Further, based on our bank-level innovation, we further present a hybrid stationary dataflow to achieve a pipelined processing to hide the latency of the SDDMM stage and the SpMM stage. The evaluation result shows that our design achieves up to $12.36\times$ performance improvement compared with the conventional NVPIM architecture with an up to $3.4\times$ energy efficiency improvement. In addition, our design also achieves up to $8.6\times$ energy efficiency improvement over the non-PIM design without sacrificing the processing latency.

II. Backgrounds

A. Sparse Attention in Transformer

The *transformer* model utilizes self-attention, where the input sequence of token embeddings X_{in} is first converted into a queries (Q), keys (K) and value (V) matrices by multiplying them with three

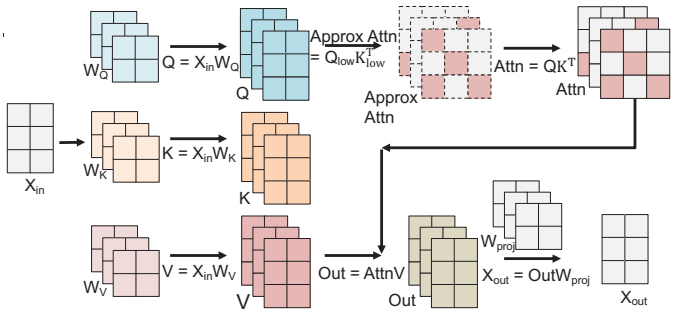

Fig. 2. The computation process of a typical sparse attention block.

weight matrices W^Q, W^K and W^V. The output is computed as a weighted sum of the values, where the weight of each value is computed by a similarity function of the query with the corresponding key. We compute the matrix of outputs using the scaled dot-product attention layer:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \quad (1)$$

Here, d_k is a pre-defined constant and $QK^T/\sqrt{d_k}$ is also called *attention map*.

Researchers attempted to reduce the computation related to the attention map by exploiting the sparsity [7]. By eliminating unimportant attention scores, sparse attention could maintain the important correlation between tokens while significantly reducing computational complexity. With a sparse attention map, matrix multiplication between Q and K^T is converted into sampled dense-dense matrix multiplication (SDDMM) while the weighted sum of values turns into a sparse-dense matrix multiplication (SpMM). The computation process of a sparse attention block is illustrated in Fig. 2. One representative approach proposed in [4] adopts a precision gating method to generate the attention mask. An approximated attention map is computed by performing dense multiplication with low-precision Q and K matrices. Then, the attention mask is generated by comparing the approximated attention map with a predefined threshold. In addition to precision gating, [5] adopts a low-rank decomposition method to generate the approximated attention map.

B. NVPIM Basic

We select ReRAM as our target device because of its high density and energy efficiency, as shown in previous designs for other neural networks [8], [9]. Typically, the memory cells are organized into arrays with the crossbar structure. The PIM array supports VMM as the computation primitive. The elements of the matrix are stored in the memory array as the conductance of the memory cells. The vector is first fed into the input register of the bank. Then, each element of the vector is converted into the voltage generated by the wordline driver and applied to the corresponding wordlines of the array. The accumulation result can be represented as the current on the bitline. The current is then converted to digital domain by an analog digital converter (ADC) or multi-bit sense amplifier (MBSA) for the subsequent process at the local PE. The final results are stored in the output register for further accumulation or external access.

However, in a real NVPIM design, the input parallelism is determined by the number of wordlines that can be turned on in parallel, while turning on too many wordlines will lead to non-negligible errors in the accumulation current. The output parallelism is bounded by the maximum number of ADCs that can fit into the area and power budget of the design. Thus, in state-of-the-art designs [10], the memory array size is 512×512. For each cycle, only 8 wordlines are turned on, while only 64 bitlines are sensed in parallel instead of activating the whole array simultaneously.

C. Challenges

For one NVPIM bank, the sparse computation pattern leads to low utilization rate, which originates from rigid input/output dimensions of the VMM primitive. We can abstract each PIM bank into a $M \times N$ PE array which computes the inner product between a $M \times N$ matrix and a $1 \times M$ vector in each cycle. As shown in Fig. 3(a), in the attention map computation stage, the k vectors are stored in the memory array, and the q vectors are streamed into the NVPIM bank, where the output attention map matrix is streamed

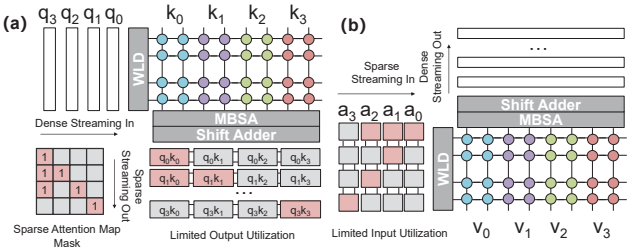

Fig. 3. An example to illustrate the inefficiency to map the (a) attention computation stage and (b) output computation stage to the conventional analog PIM bank.

out. At each cycle, M elements in a q vector can be computed with N k vectors. In this case, the NVPIM bank is fully utilized only when N continuous indexes in the attention map mask is valid (attnMask[i][j:j+N] is all '1'). In another word, the output dimension of the NVPIM bank cannot be fully filled, which will reduce the effective output parallelism. Similarly, for the output computation stage (shown in Fig. 3(b)), the V vectors are stored in the memory array and the sparse attention map matrix is streamed into the NVPIM bank. At each cycle, M elements in a row of the attention map are computed with N V vectors. We need to ensure N continuous indexes in the attention map mask are valid (attnMask[i][j:j+N] is all '1') to fully utilize the PE array. In another word, the input dimension is not fully filled with a reduction of the effective input parallelism. In general, for current NVPIM architecture, the computation primitive provided by the NVPIM bank is not suitable for sparse processing in the attention model.

III. Reconfigurable PIM Bank

A. Design Principle

To improve the mapping efficiency of the sparse workloads, the basic idea is to collapse the input or output dimensions according to the presence of sparsity. Mapping the sparse output dimension in the attention map computation stage into a temporal loop will improve the mapping efficiency. For each computation cycle, we compute one non-zero value in the sparse attention map from the inner product between one q and k vectors (attnMask[i][j]==1). The computation corresponding to the pruned attention scores can be skipped. Similarly, when the input dimension is sparse, the input is viewed as a scalar and each computation cycle produces one output vector corresponding to the product of the non-zero attention map and one v vector. For each computation cycle, we feed one non-zero value in the sparse attention map as a scalar and perform a multiplication between the scalar and one specific v vector. The computation corresponding to the zero-value attention map can be skipped.

B. Implementation

Considering the optimized data mapping scheme, we need to re-design the NVPIM bank to support two additional vector-based primitves, scalar-vector multiplication (SVM) and inner-product (IP) mode. Fig. 4 shows the proposed reconfigurable NVPIM bank architecture and the detailed data path, inspired by [11], [12]. Our design is based on the conventional analog PIM bank as shown in Fig. 4 (a). The proposed PIM bank contains an I/O register (IR/OR), wordline drivers (WLD), one or multiple memory array(s), read out circuit (including column multiplex (Col.MUXs) and MBSAs), and a local processing element (PE). We denote the matrix ($M \times N$) stored in the memory array as matrix A, the input vector ($M \times 1$) as vector B, and we assume that the precision of both A and B are 8-bit. Matrix A is assigned to the corresponding M rows and $N \times 8$ columns in

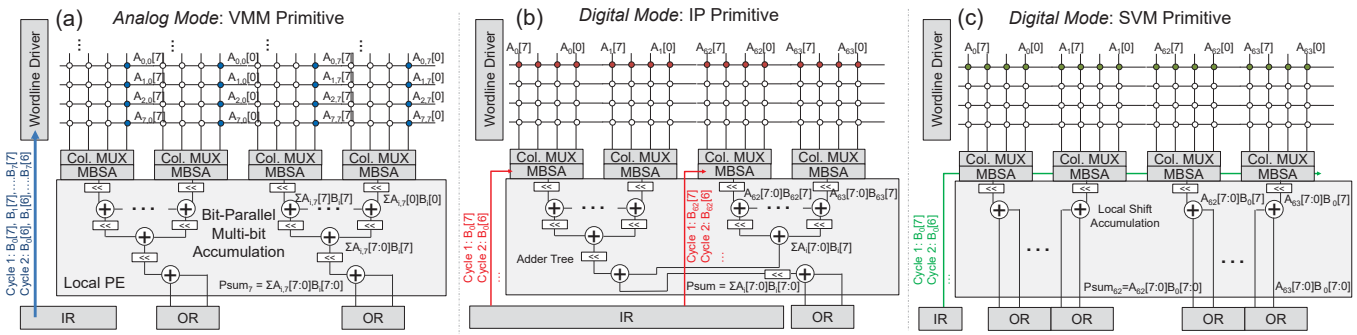

Fig. 4. Illustration of the computation path of the PIM bank in (a) analog mode for VMM primitive, (b) digital mode for IP primitive and (c) digital mode for SVM primitive.

the memory array. The bits for one element in A is interleaved to different columns with a stride of 8 to ensure the parallel computation. In each cycle, one bit of each element in B is fetched from the input register to the wordline driver that activates M wordlines in parallel to take in these M 1-bit inputs. The current at each bitline will represent a bit-wise multiplication and accumulation (MAC) result between B and one column of A. Along each bitline, one MBSA converts the bitline current to a multi-bit digital value. Each MBSA requires M levels, and each MBSA contains can be implemented by M sense amplifiers (SAs) with M different reference levels, as shown in Fig. 5 (a). For every 8 MBSAs, the bit-wise MAC results are further shifted and accumulated through an 8-to-1 adder tree and then sent to the shift-accumulator collecting the partial sum. The Col.Mul is set to select a specific bitline to the MBSA, to implement the stride-8 bit-interleaved scheme. A thermal-to-binary decoder is needed to generate a data with binary format for further processing.

Our design reuses the available resources in the conventional analog PIM bank to minimize the area overhead. The computing paradigm of the IP primitive is shown in Fig. 4 (b). The key to support IP primitive is to configure the MBSA into multiple SAs by providing same reference voltage to each SA, as shown in Fig. 5 (b) (At the same time, the thermal-to-binary decoder is bypassed). The Col.Mux is set to connect 8 bitlines to select multiple bitlines in parallel to multiple SAs. Here, we denote the vector stored in the memory array as vector A, and the input vector as vector B. Both vectors have the shape of $MN \times 1$ (instead of $M \times 1$ to keep the throughput). Each bit in one element of A is directly stored in 8 consecutive bitlines ensure the parallel read. Before the computation starts, the vector A is first read out by activating one specific wordline. In this case, $N \times 8$ M-level MBSAs can be used to sense $N \times 8 \times M$ bitlines in parallel. We can read out vector A with length of MN, where each element in the memory array is 8 bit. We still feed the vector B bit-by-bit to the MBSA. $M \times N$ element-wise multiplication results can be generated by performing an AND operation between the sensed vector A and each bit of vector B. Then we configure the local PE to an MN-to-1 adder tree to generate the IP result. We still

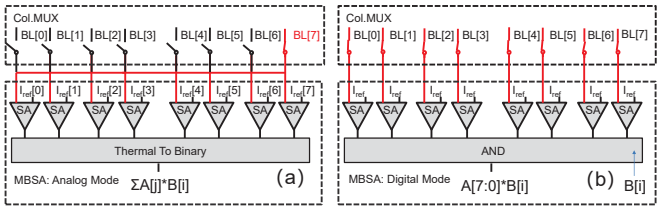

Fig. 5. The data path of Col.MUX and MBSA in (a) analog mode for VMM primitive and (b) digital mode for IP and SVM primitive.

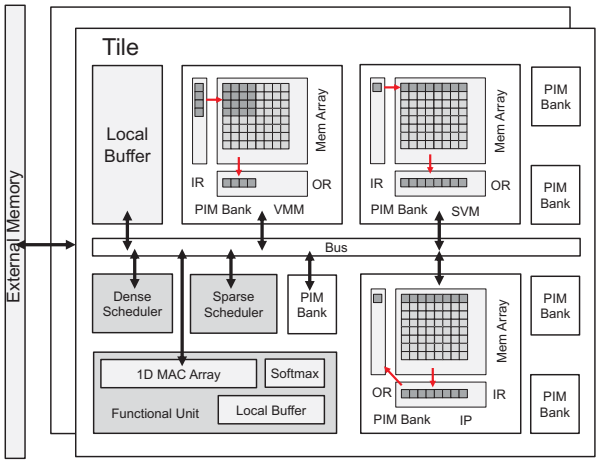

Fig. 6. The overall architecture of proposed design.

need one shift-accumulator to accumulate the bit-wise MAC results corresponding to different input bits. The computing paradigm of the SVM primitive is similar to the IP primitive, as shown in Fig. 4 (c). Here, we denote the vector stored in the memory array as vector A and the input scalar as B. The MBSA is also configured into multiple SAs and a vector A with length MN is read out. The scalar B is broadcast bit-by-bit to the all MBSAs, and then an AND operation is performed to generate MN 8b element-wise multiplication results. Then we configure the local PE to MN shift-accumulator to generate the SVM results.

It is worth noting that the vector primitives in our proposed NVPIM bank does not induce throughput reduction comparing with the analog VMM primitives. Despite the row activated in parallel is reduced $8\times$, $8\times$ columns can be sensed in parallel as the MBSA is configured into multiple SAs.

IV. Accelerator Architecture

A. Architecture Overview

Fig. 6 depicts the overall architecture of the proposed design. We organize the aforementioned reconfigurable PIM bank into tiles. Each tile can communicate with the external memory or the other tiles through a shared interface. Within each tile, apart from the reconfigurable PIM banks, we also built a local buffer, a functional unit, and dense/sparse schedulers. The sparse attention computation can be decomposed into multiple dense and sparse matrix computation stages. The PIM banks in the tile are used for both dense and sparse matrix computation. The analog mode is used to perform a VMM for the dense matrix operation, and the digital modes are used to perform IP or SVM for the sparse matrix computation. The functional unit is

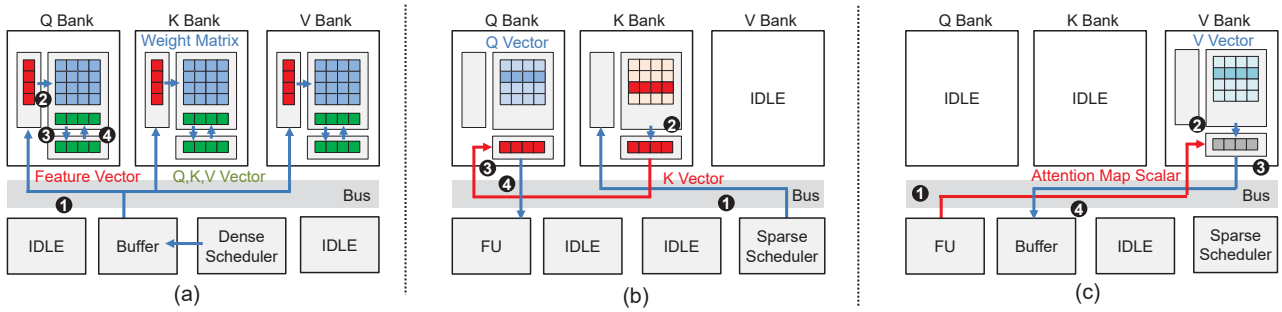

Fig. 7. An example of data mapping and computation flow for (a) dense operation, (b) $Q \times K^T$ and (c) Attn $\times V$.

used to perform softmax and normalization operations. Intermediate data, such as embeddings (X), are stored in the local buffer. Finally, a dense and a sparse scheduler are used to orchestrate the computation flow by moving the data and assigning the computation task to each PIM bank.

Our design features a hybrid stationary dataflow that distributes the computation of different stages into different PIM banks to fully utilize the computation resources. Moreover, we could enable a dedicated pipelined execution scheme among different stages to reduce processing latency.

B. Computation Flow

We would like to use a simple example to demonstrate how our proposed design executes both dense and sparse computation in the sparse attention operation.

1) Dense Computation: The dense operation includes the QKV generation layer, the prediction of the attention map, the projection layer, and the FFN layer (linear layers). We use the QKV generation layer as an example. Before starting the calculation, we divide the PIM banks into Q, K, V bank, and the weight matrices (W_q,W_k,Wv) are assigned to the corresponding bank. The computation is performed in token order. According to the token index, an input feature vector is fetched from the local buffer and sent to all Q, K, V PIM banks. Then, all Q, K, V banks are configured into the VMM mode to perform a VMM computation. The dense scheduler triggers Q, K, V banks to perform the VMM between the input feature vector and the weight matrices, and the results are temporarily stored at the output register inside each PIM bank. At the end of the dense computation, the results are directly stored back to the memory array in the PIM bank for further sparse computation.

2) Attention Map Computation: Each output attention score is the result of a dot-product of a pair of query and key vectors. After the QKV generation layer, the Q and K matrices are stored at the Q banks and K banks. We use a "Q stationary" dataflow, where the Q banks are used as the computation bank and the K banks as the memory bank. The mask of the sparse attention map is stored at the sparse scheduler. According to indices of K vectors corresponding to non-zero attention score, the sparse scheduler issues the memory access requests to the K bank. The K vector will be gathered to the sparse scheduler and further scattered to the corresponding Q banks. Then, the Q banks are configured into the IP mode, and the sparse scheduler triggers the computation of the corresponding Q bank. According to the Q index, a specific Q vector is selected in the memory array and a vector-vector IP between the Q vector and the K vector is computed to generate one attention score. The generated results are further sent back to the function unit for softmax, and temporarily stored in the attention map buffer in the function unit.

3) Output Computation: We obtain the result of the attention operation by multiplying the attention map with the V matrix in this stage. To reduce the data movement, we use a 'V stationary' dataflow. According to the index of V vectors corresponding to non-zero attention scores, the sparse scheduler fetches the attention map from the function unit to the V banks. Then, the V banks are configured into the SVM mode, and the sparse scheduler triggers the computation. According to the non-zero index of the attention map, a specific v vector is selected in the memory array, and a scalar-vector multiplication is computed. Then, we iterate over all attention score to generates the final output features which are further sent back to the local buffer.

C. Pipeline Scheme

In the attention map computation stage, the Q and K banks are used for data access or computation while the V banks are idle. In contrast, in the output computation stage, only V bank is used for data access or computation. Thus, there is an opportunity to form a pipeline to hide the latency. Fig. 8 shows the pipeline scheme in our design. The pipeline contains 2 stages, including the attention map computation and the output computation. We group multiple rows in the attention map into one chunk, and split the attention map into multiple chunks. We first trigger the attention map computation of the first chunk in the attention map. When the attention map of the first chunk is generated, we trigger the output computation of the first chunk and the attention map computation of the second chunk at the same time. Two individual controllers are designed inside the sparse scheduler to manage the computation flow for attention map computation and output computation. The computation of the subsequent stage is triggered only when both the attention map computation and the output computation of the previous stage are completed. Other than latency reduction, a chunk level pipeline could also reduce the attention map storage requirement. Instead of buffering the whole attention map, we only need to buffer the attention map for two chunks.

V. Evaluation Methodology

A. Workload Setup

We evaluate our design on 3 vision *Transformer* models and 1 NLP *Transformer* model. For vision tasks, we use 6 models including DeiT (DeiT Tiny, DeiT Small) [13], PVT (PVT Tiny, PVT Small) [14] and PiT (PiT Tiny) [15] on ImageNet [16] dataset, where the input size is $3 \times 224 \times 224$ and the patch size is 16×16. For the language tasks, we use BERT [2] for GLUE [17], SQuAD v1.1 [18], and CLOTH [19] benchmarks. We use the precision gating method proposed in [4] to generate the attention map mask. The other layer such as linear layer and FFN layers are quantized to 8 bit for both activation and weights. Each vision model is re-trained 60 epochs after quantization and sparsification, to recover the accuracy loss caused by the gating. The language models are fine-tuned on each downstream task based on pretrained BERT model until converge.

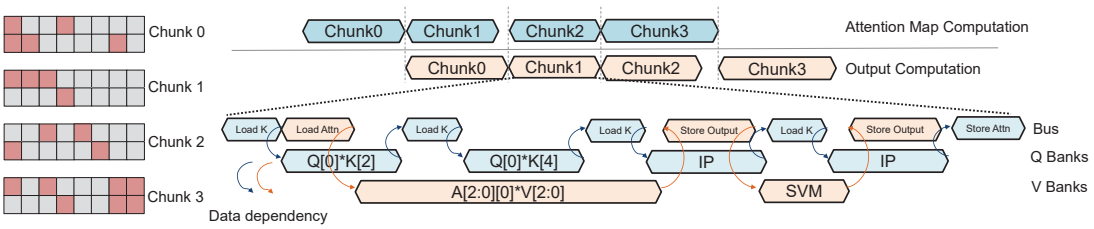

Fig. 8. Pipelined execution scheme to hide the latency.

B. Hardware Setup

For the NVPIM bank, we use the memory array model from [9]. The energy and area for the adder in local PE is extracted from synthesized results with 28 nm standard cell library. For the analog component such as MBSA, we implement the circuit in transistor level and extract the area and energy consumption from the schematics. The NVPIM bank level area breakdown is shown in Table I.

The hardware configuration is shown in Table II. The results of the function unit is extracted from [5], and the results of the local buffer is extracted from CACTI. Each PIM Bank could deliver about 9.14 GOPS throughput with 128 kB memory capacity. We set 72 banks in each tile, which could deliver 658 GOPS and 9 MB memory capacity. Based on bank-level results, we further implement a cycle-accurate simulator to capture the total performance and energy consumption for each workload.

C. Baseline Setup

For the baseline NVPIM accelerators, we quantitatively compare our design with ReTransformer [3], as implemented with our cycle-accurate simulator. ReTransformer is designed for dense attention computation, so we apply the method proposed in SRE [6] to support the sparse attention. We also compare our design with two non-PIM based designs. To make it fair, we set the memory access energy/latency and the computation energy/latency to be same as the NVPIM based designs. Then, we select a proper design budget so that both non-PIM and NVPIM designs could deliver the same peak throughput. Our assumption ensures the results from non-PIM design are more optimistic.

We choose DOTA [5] and Sanger [4] as they use similar sparse algorithm. For these non-PIM based designs, we develop a behavior model to evaluate the latency and energy cost. For DOTA, we scale down the number of lanes in the original design (2TOPS) to match the peak throughput with our baseline non-PIM accelerator (512GOPS), while for Sanger, we directly choose the original hardware configuration since it delivers similar effective throughput (529GOPS).

Component	Params.	Spec.	Area(mm^2)	
			Con.	Ours
Memory Array	Size	512×512	0.0128	0.0128
	Number	4		
Local PE	No. of Adders	64	0.0078	0.015
Input Register	Size	2K Byte	0.0021	0.0021
Output Register	Size	256 Byte	0.00077	0.00077
8-level MBSA	Number	64	0.01	0.01
Total	–	–	0.033	0.041

TABLE I
AREA BREAKDOWN FOR OUR NVPIM BANK AND THE CONVENTIONAL NVPIM BANK.

Component	Params	Spec
NVPIM Bank	Throughput	64 MACs/14 ns
	Comp.Energy	93 fJ/MAC
	Capacity	128 kB
	Mem. Energy	2.3 pJ/Byte
	Number	72
Function Unit	Throughput	64 MACs/ cycle
	Comp.Energy	113 fJ/MAC
Local Buffer	Capacity	512 kB
	Mem.Energy	4.5 pJ/Byte
Local Bus	Bandwidth	512 Byte/cycle

TABLE II
HARDWARE CONFIGURATIONS

VI. Results and Discussion

A. Main Results

We separately depict the sparsity and accuracy result as grey dots in Fig. 9. For most of the models, the sparse attention algorithm incurs negligible accuracy loss ($< 1\%$) while archiving a relatively high sparsity ratio in attention score ($> 85\%$). Sparse attention could even improve the accuracy on simple downstream tasks like SST-2 and CLOTH, since the pruning process reduce the overfitting of the baseline model. We illustrate the improvement of our proposed design over the baseline NVPIM accelerator in Fig. 9. The result shows that our proposed design could achieve a speedup ranging from $2.95\times$ to $12.36\times$ over the baseline design. In terms of energy, our design could achieve a $1.2\times$ to $3.4\times$ energy efficiency improvement comparing with the baseline accelerator, as shown in Fig. 9. The energy efficiency improvement comes from the improvement of the bank-level utilization rate. Similar as the speedup, we also observe a larger energy efficiency improvement when the sparsity level increases.

B. Scalability Analysis

We investigate the bank-level scalability when we have more available banks to improve the token-level parallelism for both attention map computation stage and output computation stage. The result is shown in Fig. 10(a). The processing latency can be reduced

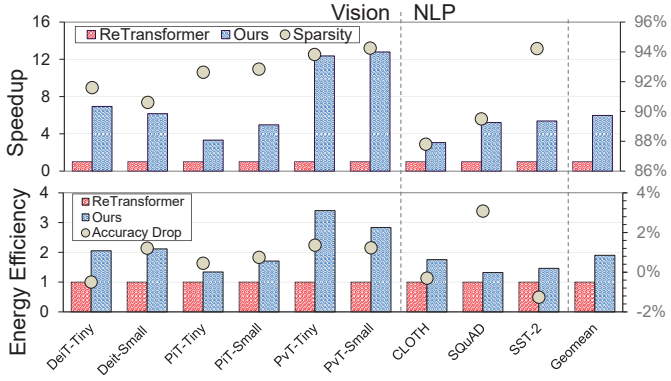

Fig. 9. Speedup and the energy efficiency results on vision and NLP tasks.

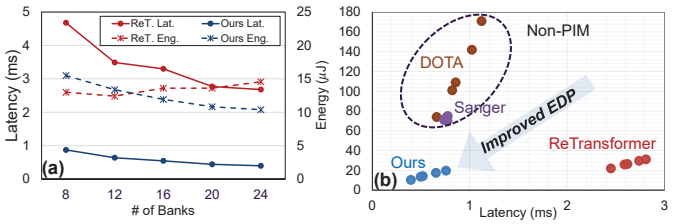

Fig. 10. (a) Energy-latency trade-off. (b) The latency and energy results with different number of banks for our design and ReTransformer.

about 50% when we increase the bank-level parallelism from 8 to 24, while the latency reduction is only 30% for the baseline design. Moreover, the gap between output design and baseline expands with more banks. The results also show a reduction of energy consumption when we use more banks in the proposed design, while the baseline design consumes more energy. When we increase the token-level parallelism, the token reuse rate can be increased by reusing the K vector for different Q vectors. Thus, we could achieve a lower energy consumption in terms of memory access cost. In contrast, the baseline accelerator consumes more energy in the computation as the inter-bank efficiency is further reduced by mapping the Q, K, V matrices into more banks.

C. Comparison with Non-PIM Accelerators

The energy-latency trade-off of both PIM and non-PIM accelerator is shown in Fig. 10 (b). Generally, the non-PIM designs could achieve lower latency than ReTransformer, while the NVPIM-based designs could achieve a higher energy efficiency. In contrast, our proposed design could achieve a better energy-latency trade-off, where both energy consumption and latency is relatively low. For example, our design could achieve up to $3.8\times$ and $8.6\times$ energy reduction comparing with Sanger and DOTA, respectively, while the processing latency is in the same level. However, Retransformer could deliver $3.71\times$ and $8.5\times$ energy reduction at the cost of $4.8\times$ and $3.3\times$ latency overhead, respectively.

To further understand the energy efficiency improvement, we present an energy consumption breakdown of our design, ReTransformer and non-PIM designs. Fig. 11 shows the energy breakdown of four designs to process a DeiT-Small model with different sparsity levels. For all sparsity levels, our design consumes less energy on computation. The lower computation energy comes from a higher array utilization rate. However, our design requires higher memory access energy due to smaller data reuse opportunities while using a vector-based processing scheme.

Comparing with Sanger, our design could achieve both computation and memory access energy reduction. The computation energy reduction comes from the higher utilization rate in our design. The memory access energy reduction comes from our hybrid stationary dataflow, where the data movement of Q and V is reduced. Comparing with DOTA, our design achieves a similar computation energy with a large reduction of memory access energy. DOTA uses 1-D SIMD-based processing scheme and all the Q, K, V vectors require to be fetched multiple times during the computation.

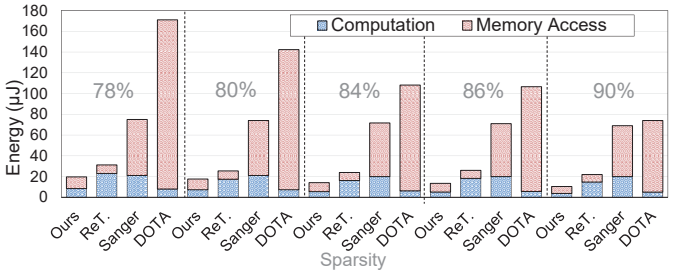

Fig. 11. Energy breakdown for PIM and non-PIM designs with different sparsity level on DeiT-Small model.

VII. Conclusion

In this work, we propose a NVPIM architecture, which aims at solving the challenge of the sparse attention on NVPIM architecture with a dynamic and unstructured pattern. We first propose a reconfigurable NVPIM bank with vector-based primitives to improve the intra-bank utilization rate for the SDDMM and SpMM computation. Based on our bank-level innovation, we design a hybrid stationary dataflow which enables a pipelined processing scheme to hide the computation latency of the SDDMM stage and SpMM stage. Our proposed design could achieve up to $12.36\times$ performance improvement over conventional NVPIM architecture (ReTransformer), while delivering a up to $3.4\times$ energy efficiency improvement on a range of representative benchmarks. In addition, our proposed design could also achieve up to $8.6\times$ energy efficiency improvement over non-PIM design without sacrificing the processing latency.

Acknowledgement

This work was supported by National Science Foundation under grant CNS-2233808, EECS-2023752, 1955246 and 2112562, and supported by Army Research Office under grant W911NF-19-2-0107.

References

[1] A. Dosovitskiy *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.
[2] J. Devlin *et al.*, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.
[3] X. Yang *et al.*, "Retransformer: Reram-based processing-in-memory architecture for transformer acceleration," in *ICCAD*, 2020.
[4] L. Lu *et al.*, "Sanger: A co-design framework for enabling sparse attention using reconfigurable architecture," in *MICRO*, 2021.
[5] Z. Qu *et al.*, "Dota: detect and omit weak attentions for scalable transformer acceleration," in *ASPLOS*, 2022.
[6] T.-H. Yang *et al.*, "Sparse reram engine: Joint exploration of activation and weight sparsity in compressed neural networks," in *ISCA*, 2019.
[7] G. M. Correia *et al.*, "Adaptively sparse transformers," *arXiv preprint arXiv:1909.00015*, 2019.
[8] P. Chi *et al.*, "PRIME: A novel processing-in-memory architecture for neural network computation in reram-based main memory," in *ISCA*, 2016.
[9] A. Shafiee *et al.*, "ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," in *ISCA*, 2016.
[10] J.-M. Hung *et al.*, "An 8-mb dc-current-free binary-to-8b precision reram nonvolatile computing-in-memory macro using time-space-readout with 1286.4-21.6 tops/w for edge-ai devices," in *ISSCC*, 2022.
[11] Q. Zheng *et al.*, "Lattice: An adc/dac-less reram-based processing-in-memory architecture for accelerating deep convolution neural networks," in *DAC*, 2020.
[12] Q. Zheng *et al.*, "Mobilatice: a depth-wise dcnn accelerator with hybrid digital/analog nonvolatile processing-in-memory block," in *ICCAD*, 2020.
[13] H. Touvron *et al.*, "Training data-efficient image transformers & distillation through attention," in *ICML*, 2021.
[14] W. Wang *et al.*, "Pyramid vision transformer: A versatile backbone for dense prediction without convolutions," in *ICCV*, 2021.
[15] B. Graham *et al.*, "Levit: a vision transformer in convnet's clothing for faster inference," in *ICCV*, 2021.
[16] J. Deng *et al.*, "ImageNet: A large-scale hierarchical image database," in *CVPR*, 2009.
[17] A. Wang *et al.*, "Glue: A multi-task benchmark and analysis platform for natural language understanding," *arXiv preprint arXiv:1804.07461*, 2018.
[18] P. Rajpurkar *et al.*, "Squad: 100,000+ questions for machine comprehension of text," *arXiv preprint arXiv:1606.05250*, 2016.
[19] Q. Xie *et al.*, "Large-scale cloze test dataset created by teachers," *arXiv preprint arXiv:1711.03225*, 2017.