SADIMM: Accelerating Sparse Attention using DIMM-based Near-memory Processing

Huize Li, *Member, IEEE,* Dan Chen✉, *Member, IEEE,* Tulika Mitra, *Member, IEEE*

***Abstract*—Self-attention mechanism is the performance bottleneck of Transformer based language models. In response, researchers have proposed sparse attention to expedite Transformer execution. However, sparse attention involves massive random access, rendering it as a memory-intensive kernel. Memory-based architectures, such as *near-memory processing* (NMP), demonstrate notable performance enhancements in memory-intensive applications. Nonetheless, existing NMP-based sparse attention accelerators face suboptimal performance due to hardware and software challenges. On the hardware front, current solutions employ homogeneous logic integration, struggling to support the diverse operations in sparse attention. On the software side, token-based dataflow is commonly adopted, leading to load imbalance after the pruning of weakly connected tokens.**

To address these challenges, this paper introduces SADIMM, a hardware-software co-designed NMP-based sparse attention accelerator. In hardware, we propose a heterogeneous integration approach to efficiently support various operations within the attention mechanism. This involves employing different logic units for different operations, thereby improving hardware efficiency. In software, we implement a dimension-based dataflow, dividing input sequences by model dimensions. This approach achieves load balancing after the pruning of weakly connected tokens. Compared to NVIDIA RTX A6000 GPU, the experimental results on BERT, BART, and GPT-2 models demonstrate that SADIMM achieves 48×, 35×, 37× speedups and 194×, 202×, 191× energy efficiency improvement, respectively.

***Index Terms*—Near-memory processing, Sparse attention accelerator, DRAM architecture, Software-hardware co-design.**

I. Introduction

Transformer-based [35] neural network models can enhance the accuracy of diverse machine learning inference, such as *natural language processing* (NLP) [3], [8] and *computer vision* (CV) [1], [10]. The accuracy enhancement is primarily attributed to the self-attention mechanism, facilitating the capture of long-term dependencies among input tokens [35]. However, these advantages come at the expense of high computational complexity to calculate the full connections among all tokens. In response to this challenge, researchers have introduced sparse attention by pruning weakly connected tokens, thereby mitigating computational complexity with a slight reduction of accuracy [30]. Nevertheless, the random distribution of non-zero values introduces additional memory access overhead, which in turn constrains system performance.

The authors are with the School of Computing, National University of Singapore, Singapore (e-mail: huizeli@nus.edu.sg, danchen@nus.edu.sg, tulika@comp.nus.edu.sg).

This research is partially supported by the National Research Foundation, Singapore under its Competitive Research Program Award NRF-CRP23-2019-0003. The correspondence of this paper should be addressed to Dan Chen.

Recently, there has been a growing interest in hardware-software co-design solutions aimed at mitigating random memory access. These solutions encompass diverse platforms, including GPU [3], [8], *Field Programmable Gate Array* (FPGA) [2], [41], and *Application Specific Integrated Circuit* (ASIC) [23], [30]. These co-design approaches outperform software-only solutions by implementing hardware-friendly scheduling and computation techniques, thereby enhancing both access and computational efficiency. Nonetheless, these computation-centric solutions still face the constraint of off-chip memory bandwidth when accelerating memory-intensive sparse attention, leading to long latency off-chip access of a substantial number of intermediate results [9].

Memory-based acceleration, which includes *processing in memory* (PIM) and *near memory processing* (NMP), holds promise for enhancing memory-intensive applications [27]. By integrating *processing elements* (PEs) in memory or utilizing the computational ability of memory, researchers propose various NMP [9], [43] and PIM-based [18], [19], [42] accelerators for attention mechanism. This paper focuses on the *Dual In-line Memory Module* (DIMM)-based NMP platform [33], which integrates PEs into commercial *dynamic random access memory* (DRAM), compatible with current main memory fabrication process while obtaining substantial parallelism and bandwidth. DIMM-based NMP platforms, renowned for their excellent performance, find widespread use in accelerating graph mining [5], [34] and recommendation systems [22], [28]. Despite their advantages, current DIMM-based NMP sparse attention accelerators face suboptimal performance due to hardware and software challenges.

In hardware, contemporary solutions employ homogeneous logic integration, struggling to support the diverse operations in sparse attention. Existing NMP-based attention accelerators assume uniform hardware resource demands for all operations within sparse attention. Consequently, they integrate identical *processing elements* (PEs) in different DIMM locations to support these operations. In practice, operations within sparse attention exhibit distinct computational and memory access requirements. A homogeneous architecture fails to address the varied access and computation needs of diverse operations in sparse attention, leading to PE idleness and diminished hardware efficiency (details in Section III-A).

In software, token-based dataflow is commonly adopted in memory-based accelerators, leading to load imbalance after the pruning of weakly connected tokens. To minimize cross-layer data transmission, contemporary NMP-based attention accelerators [9], [19], [43] adopt a token-based dataflow. Their approach divide input sequences by tokens and store distinct

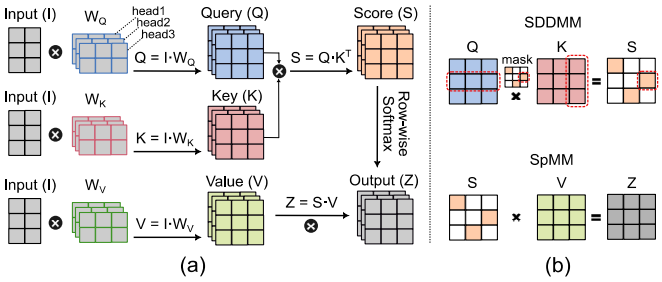

Fig. 1. (a) Multi-head attention, (b) SDDMM and SpMM

token shards in different banks. Token-based dataflow can reuse data cross different layers, thus are efficient in processing the vanilla attention, which calculates full connections among tokens [43]. However, token-based dataflow encounters challenges with sparse attention due to the irregular pruning of numerous weakly connected tokens. The irregular pruning leads to a significant load imbalance problem, as a substantial number of banks storing weakly connected tokens become idle (refer to details in Section III-B).

Given the current landscape, we present SADIMM, an innovative hardware-software co-designed NMP-based sparse attention accelerator for model inference. In hardware, SADIMM employs heterogeneous NMP architectures by integrating distinct logic units across different hierarchies of DIMM. Subsequently, different logic units are assigned to support corresponding operations, enhancing overall hardware efficiency. In software, SADIMM adopts a dimension-based dataflow, partitioning input sequences based on model dimensions. By storing identical dimensions of all tokens in the same memory bank, SADIMM can efficiently support sparse attention with balanced loads. Our contributions are as follows:

- We analyze current NMP-based sparse attention accelerators and we find that they mainly face hardware inefficiency and software load imbalance challenges.
- To improve hardware efficiency, we propose heterogeneous NMP, integrating different logic units within the memory hierarchy to support various operations.
- To address the issue of load imbalance, we propose dimension-based dataflow, dividing and storing input sequences by model dimensions to avoid PE idleness.
- The experimental results show that SADIMM achieves 156.5× (AMD Ryzen CPU), 39.6× (RTX A6000 GPU), 12.2× (ASIC-based Sanger [23]), 4.7× (NMP-based TRiM-B [28]), and 2.8× (NMP-based HAIMA [9]) speedups when comparing to modern solutions.

II. Background

A. Basics of Attention Mechanism

Transformer-based neural network models comprise multiple layers of Encoders and Decoders, with each Encoder and Decoder is structured with multi-head attention and *full connected* (FC) layers. Figure 1 (a) illustrates the computation of multi-head attention. Initially, the input sequence with n tokens is embedded into a matrix $I \in \mathbb{R}^{n \times d}$, where n denotes the sequence length and d represents the model dimension, i.e., number of features contained in each token. Subsequently,

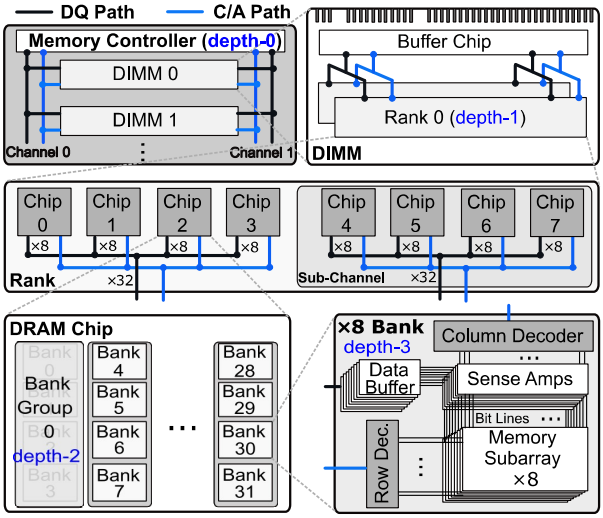

Fig. 2. DRAM-based main memory hierarchy

matrix I undergoes a *General Matrix Multiplication* (GEMM) operation (time complexity of $\mathbf{O}(nd^2)$) with three weight matrices, W_Q, W_K, and W_V, to obtain matrices Q (query), K (key), and V (value). Following this, matrix Q will undergo another GEMM operation ($\mathbf{O}(dn^2)$) with the transpose of matrix K (K^T), yielding the attention score matrix $\tilde{S} \in \mathbb{R}^{n \times n}$. Subsequently, a row-wise softmax function is applied to matrix $\tilde{S}$ to produce matrix S, which denotes the normalized probability. Finally, matrix S is subjected to a GEMM operation ($\mathbf{O}(dn^2)$) with matrix V, yielding the output matrix Z.

Both GEMM operations, $Q \times K^T$ and $S \times V$, exhibit computational complexity of $\mathbf{O}(dn^2)$, which poses challenges for vanilla attention when applied to longer sequences. Researchers observed that many tokens are weakly connected to others. Pruning these weak connections via low-precision quantization can generate sparse mask matrix to reduce computational complexity [23], [30]. With the sparse mask matrix, the GEMM operation $Q \times K^T$ is converted into *Sample Dense-Dense Matrix Multiplication* (SDDMM). Similarly, the GEMM operation $S \times V$ is transformed into *Sparse-dense Matrix Multiplication* (SpMM). Figure 1 (b) provides an example of SDDMM, taking two dense matrices, Q and K, and one sparse mask matrix M as inputs to generate the sparse matrix S. Figure 1 (b) also illustrates an example of SpMM, which takes the sparse S and dense V matrices as inputs to generate a dense output matrix Z.

B. Basics of DRAM and Near-DRAM Processing

As shown in Figure 2, DRAM-based main memory systems [14] exhibit a hierarchical structure, encompassing *channels*, *ranks*, *bank-groups*, and *banks*. At the apex of this hierarchy is the memory channel (depth-0), which includes a primary host *memory controller* (MC) and multiple secondary DRAM ranks (depth-1) connected via a DQ (data) path to the buffer chip. Each rank comprises eight DRAM chips, operating in an 8-bit lockstep manner to provide 64-bit width [14]. In DDR5 DRAM [14], a DRAM chip consists of eight bank-groups (depth-2), each housing four banks (depth-3). Each

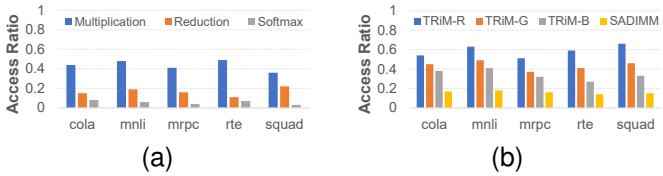

Fig. 3. (a) The ratio of memory access latency for various operations, and (b) The ratio of memory access latency for different configurations of NMP

memory bank contains multiple memory arrays to store data. Uniform *Command/Address* (C/A) signals are broadcasted to all DRAM chips, dictating their operations. Similar to the depth-1 DQ path, the depth-2 and depth-3 DQ paths are shared between a rank and eight bank-groups, and between a bank-group and four banks, respectively. While all banks in a DRAM chip operate independently, only one bank can utilize depth-1/2/3 data paths simultaneously.

Due to DRAM's hierarchical architecture, only one rank can utilize the DQ path between the MC and the rank at a time. However, if a *processing element* (PE) is integrated into the buffer chip, the depth-0 path between the MC and the buffer chip remains underutilized during computations. Consequently, all ranks can concurrently transfer data to connected PEs in buffer chips, thereby enhancing overall bandwidth. Similarly, placing a PE near the bank-group/bank can obviate the use of depth-1/2 path during computations, leading to even more bandwidth gains.

TRiM [28] provides three configurations of NMP architecture, i.e., TRiM-R, TRiM-G, and TRiM-B. TRiM-R/G/B represent integreting logical units within the rank, rank & bank-group, rank & bank-group & bank, respectively. TRiM-R costs the minimal area overhead with the minimal bandwidth and computational parallelism, which is suitable for applications where the access overhead is much higher than the computational overhead. TRiM-B has the largest area overhead and integration costs with the biggest bandwidth and computational parallelism, which is suitable for applications requiring both high bandwidth and high parallelism. Current HBM-based TransPIM [43] supports reduction with ring-wise broadcast, which transfers data between banks and introduces many cross-bank transmissions.

III. Motivation Analysis

A. Diverse Operations vs. Homogeneous PEs

Observation#1: *Sparse attention encompasses diverse operations, each characterized by unique computational and memory access demands. Existing DIMM-based NMP solutions employ uniform logic units across ranks, bank-groups, and banks. Employing a homogeneous NMP architecture to accommodate the varied operations in sparse attention results in the inefficiency of hardware resources.*

Variations in operations and their differing access requirements. In assessing the access demands of distinct operations, we focus on three primary operations within sparse attention: sparse matrix multiplication, reduction, and softmax. The generation of sparse mask matrices employs the pruning algorithm of Sanger [23]. Our experimentation involves the execution of five datasets [36], i.e., cola, mnli, mrpc, rte, and squad, on a DIMM-based NMP accelerator, maintaining consistent configurations with TRiM-G [28]. We also configure a softmax unit in each NMP PE. The detailed methodology is shown in Section VI. Figure 3 (a) illustrates the proportion of memory access latency to the total latency for the identified operations. The latency caused by memory access means the average PEs' stall time since waiting for data. Assuming we have N PEs and the overall latency of the NMP accelerator is T. The execution time and stall time of PE_i is t_i and s_i ($T = s_i + t_i$), respectively. The average stall ratio for all PEs is $\frac{\sum_{i=1}^{N} s_i}{NT}$. Notably, sparse matrix multiplication incurs over 40% memory access latency, primarily attributed to the substantial random accesses involved in sparse matrices. Conversely, reduction and softmax operations exhibit lower memory access latency percentages, approximately 10% and 5%, respectively, as these operations do not necessitate extensive access to sparse matrices.

Impact of NMP configuration on parallelism and bandwidth. To evaluate the influence of varied NMP configurations, we set up three DIMM-based NMP hardware in alignment with TRiM [28], namely TRiM-R, TRiM-G, and TRiM-B. In Figure 3 (b), we present the proportion of memory access latency to the total latency in sparse attention. Notably, TRiM-B exhibits a reduced percentage of memory access latency compared to TRiM-R, attributed to enhanced bandwidth achieved by avoiding the depth-1/2 DQ paths. Beyond bandwidth variations, distinct computational parallelism is evident. Specifically, near-bank NMP incorporates a compact PE area within each bank, making it adept at parallel processing small data batches. Conversely, near-rank NMP integrates more expansive PEs within buffer chip, providing heightened computing capability for processing aggregated data.

Challenges in utilizing current DIMM-based NMP platforms for sparse attention. DIMM-based NMP platforms are widely applied in graph mining [5], [34] and recommendation systems [22], [28]. Graph mining accelerators primarily focus on *sparse matrix-vector multiplication* (SpMV), while recommendation system accelerators center around *gather and reduction* (GnR). SpMV or GnR relies on a single core operation and, as a result, follow a uniform memory access pattern. For example, TRiM, a well-known GnR accelerator, performs gather-and-reduction operations to extract hundreds of elements from billions. After completing the reduction, TRiM uses the same PE to execute SpMV operations on the reduced set of elements. Since the SpMV kernel requires minimal computation, a homogeneous PE with a consistent access pattern suffices to handle the task efficiently. Consequently, existing DIMM-based NMP solutions are homogeneously integrated to align with their specific core operations. In contrast, sparse attention workloads involve a variety of core operations, which demand greater computational resources and exhibit diverse memory access patterns, as analyzed in Observation#1.

Applying current DIMM-based NMP accelerators to support the varied operations in sparse attention introduces two inherent challenges. *First, equitably distributing tasks to near-memory PEs amplifies access overhead.* Compared to near-

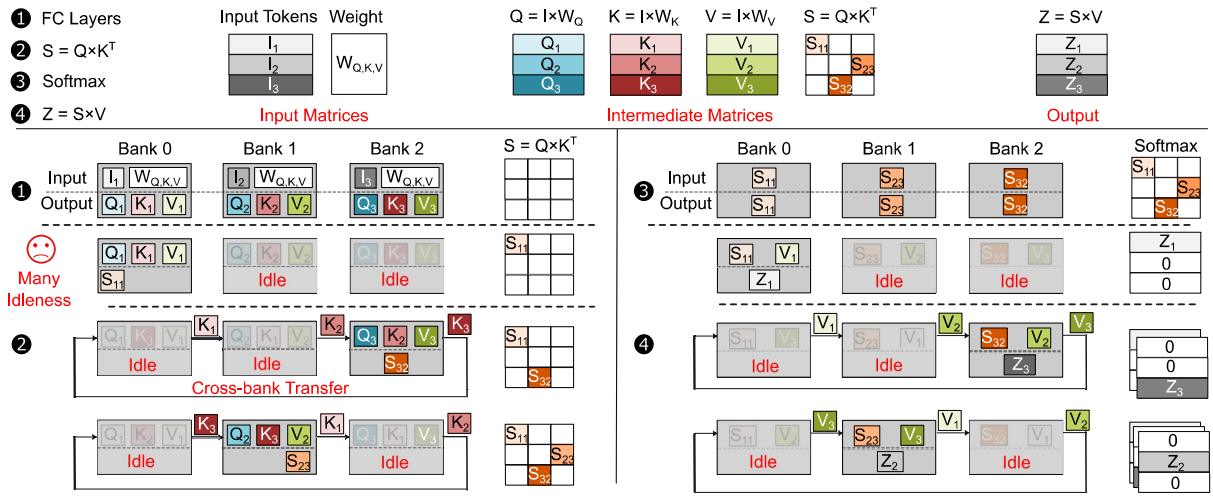

Fig. 4. Token-based [43] sparse attention dataflow causes load imbalance problems.

rank PEs, near-bank PEs are in closer proximity to the memory array, resulting in lower access latency to the data stored in memory banks. Conversely, near-rank PEs, being closer to the memory controller, are better suited for aggregating intermediate results. Assigning access-demanding operations to near-rank PEs introduces more access overheads than near-bank PEs. *Secondly, homogeneous integration leads to increased area overhead and diminished energy efficiency.* NMP-based platforms are designed to off-load memory-intensive atomic operations to memory. Therefore, a large number of PEs may only need to support a small number of atomic operations. Integrating more powerful PEs equally to support diverse operations may render some logic units unused, which will waste area and reduce energy efficiency.

B. Regular Mapping vs. Irregular Pruning

Observation#2: *To mitigate cross-layer data transfers, contemporary NMP-based attention accelerators adopt a token-based dataflow, evenly distributing input tokens across different banks. However, the adoption of sparse attention, aimed at reducing computational complexity by randomly pruning weakly connected tokens, introduces challenges related to load imbalance and idle PEs.*

Layer-based dataflow. In pursuit of enhanced parallelism, prevalent NMP-based *deep neural network* (DNN) accelerators [13] employ a layer-based dataflow, allocating sufficient memory resources to parallelize computations for different output elements in a layer. This approach dedicates significant memory resources to concurrently process computations for distinct outputs within a layer, requiring the complete loading of data before processing each layer. However, as indicated by prior research [43], approximately 60% of the execution time is consumed by data movement in the layer-based dataflow. The prolonged data movement time stems from two primary factors [43]. First, for optimal parallelism, the layer-based dataflow necessitates data duplication in memory for parallel computations, leading to an increased volume of loaded data. Second, many parallel data layouts overlook the exploitation of data reuse between adjacent layers, compelling the loading of all data for attention layers.

Token-based dataflow. To mitigate substantial cross-layer data transmissions, contemporary NMP-based attention accelerators [9], [19], [20], [43] adopt a token-based dataflow. This strategy involves partitioning input tokens into different shards and assigning these shards to distinct memory banks. During attention computation, each memory bank independently processes its associated token shard across various layers. Due to the significantly enhanced data reuse across layers, token-based dataflow reduces much of the data movement overhead in comparison to layer-based dataflow. Figure 4 illustrates an instance of token-based dataflow in sparse attention, containing *full connected* (FC) layer, SDDMM operation with $S = Q \times K^{\mathsf{T}}$, row-wise softmax, and SpMM operation with $Z = S \times V$. The SDDMM and SpMM operations in this example correspond to those in Figure 1 (b). Specifically, $L = D = N = 3$, where L represents sequence length, D denotes model dimension, and N signifies the number of banks. All matrices are token-divided, with row vectors depicted at the top of Figure 4.

Irregular pruning and load imbalance. In step ❶, each bank accommodates $\frac{L}{N}$ tokens, each represented as a vector with D dimensions. The GEMM operations in FC layers produce three $\frac{L}{N} \times D$ sub-matrices in each bank, denoted as Q_i, K_i, and V_i ($1 \le i \le N$). Progressing to step ❷, each bank initiates the matrix multiplication $S_{ii} = Q_i \times K_i^{\mathsf{T}}$. Subsequently, a ring-wise token transfers convey K_i to bank$_j$ for computing $S_{ji} = Q_j \times K_i^{\mathsf{T}}$ ($j = i+1$ in our example). Some banks experience idleness due to token pruning via sparse mask matrices, such as bank$_1$ with weakly connected tokens Q_2 and K_2. Step ❸ involves row-wise softmax with the matrix S. Finally, in step ❹, the SpMM operation $Z = S \times V$ will also introduce PE idleness, which is caused by the random distribution of a large number of non-zeros in matrix S. The irregular pruning of weak connected tokens in sparse attention leads to varying sparse patterns with strong randomness, making it challenging to address load imbalance using an on-chip scheduler [23] due to the significant random scheduling overhead it introduces.

C. Key Ideas of SADIMM

Hardware acceleration: To effectively accommodate

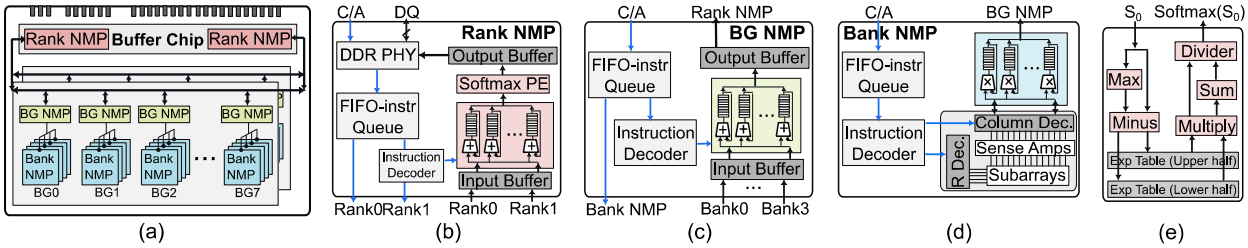

Fig. 5. (a) Overall architecture of SADIMM, (b) Near-rank NMP architecture, (c) Near bank-group NMP architecture, (d) Near-bank NMP architecture, and (e) Architecture of softmax unit

the diverse operations within sparse attention mechanism, SADIMM introduces subtle enhancements to the conventional DIMM architecture, enabling the acceleration of multiple operations inherent to sparse attention. Specifically, a heterogeneous integration strategy is employed, incorporating diverse logic units into the memory hierarchy, i.e., near-rank PEs, near bank-group PEs, and near-bank PEs. These logic units are then allocated to specific operations based on their unique access requirements (as analysed in Section III-A). Significantly, this heterogeneous integration approach not only optimizes area utilization but also enhances energy efficiency.

Dataflow: SADIMM employs an innovative dataflow paradigm that aligns sparse attention computation with a memory-based architecture using a dimension-based sharding mechanism. This approach entails partitioning input sequences based on model dimensions and assigning these dimensions to distinct memory partitions. Throughout acceleration, each memory partition independently processes its associated dimension shard across different layers. Observation#2 emphasizes that token-based dataflow introduces load imbalance to sparse attention, resulting in significant random scheduling overhead to balance loads. In contrast, the dimension-based dataflow facilitates a load-balanced distribution in memory banks after irregular pruning.

IV. SADIMM Hardware Acceleration

A. Overall Architecture

The overall architecture of SADIMM is depicted in Figure 5 (a). Derived from the commercial *Load-Reduced DIMMs* (LRDIMM) [14], we implement specific modifications to enhance its compatibility with diverse operations in sparse attention mechanism. The integration encompasses three types of NMP: near-rank PEs, near bank-group PEs, and near-bank PEs. The buffer chip incorporates two near-rank PEs responsible for handling input/output data to/from two ranks. Each bank-group in the rank pairs with one PE to facilitate near bank-group NMP, ensuring bandwidth gains at the bank-group level. All banks within the bank-group integrate with PEs to maximize bank-level bandwidth. Each near-bank PE operates in two access modes for obtaining data from all banks: local access and global access. Local access denotes a near-bank PE accessing its integrated memory bank, such as PE_i accessing bank_i. Global access signifies that PE_0 is requesting data from banks other than bank_0. The latency of global access is notably higher than local access due to additional broadcasts [43]. Importantly, the modified DIMM retains the capability to function as a conventional main memory device by preserving all conventional interfaces. To utilize the advantages of DRAM's lockstep access, we configure the host processor to access one token at a time.

As depicted in Figure 1 and detailed in Section III-A, sparse attention contains three fundamental operations: sparse matrix multiplication, reduction, and softmax. These operations entail three operators, i.e., multiplication, addition, and exponentiation. Consequently, the integration of SADIMM necessitates the inclusion of multipliers, adders, and exponential lookup tables. The critical consideration is determining where these logic units should be integrated to satisfy the varied demands of different operations. Figure 3 (a) presents the access characteristics of diverse operations. Specifically, sparse matrix multiplication has the highest memory access requirements, suggesting that utilizing near-bank PEs for processing matrix multiplication incurs minimal access overhead. Conversely, the reduction operation involves summing up intermediate results produced by each bank, which does not involve random memory access. To fully leverage the hierarchy of DIMM, transferring intermediate results from each bank to a bank-group and processing them with near bank-group PEs is optimal. The rank-level PE, situated at the memory's root, is also well-suited for processing the reduction operation. The above configuration will generate intermediate results in banks and bank-groups. The rank-level NMP will accumulate these intermediate results to generate S matrix. To avoid data transfers, we integrate a softmax unit in the near-rank PE.

B. Integration Details

Rank-level NMP. The hardware configuration of the rank-level NMP in SADIMM is depicted in Figure 5 (b). We integrate a DDR PHY and protocol engine, mirroring the design of a conventional DIMM buffer chip that relays the DRAM *Command/Address* (C/A) and *Data* (DQ) signals to and from the host-side memory controller. Central to this architecture is the *First-In-First-Out* (FIFO) instruction queue, responsible for receiving instructions from the host-side memory controller. These instructions are subsequently parsed to access data and control the near-rank PEs. Given the near-rank PE's primary role in reduction and softmax operations, the rank-level NMP includes 32-bit floating-point accumulators and a softmax unit. The input/output (16B/32B) buffer efficiently manages the reception and storage of data from/to all bank-groups in the ranks, respectively. In line with ReCross [22], a 2-bit flag `nmp_level` $= 01$ distinguishes which instructions should be executed by the near-rank NMP.

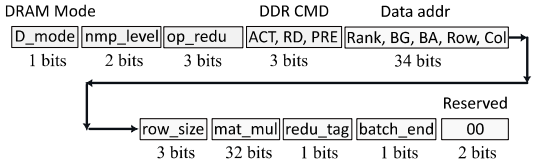

Fig. 6. The instruction format of SADIMM

Bank-group level NMP. As shown in Figure 5 (c), the near bank-group NMP configuration mirrors the near-rank NMP in its task of performing the reduction operation. Hence, SADIMM integrates 32-bit floating-point accumulators within each bank-group to aggregate data from different banks. We configure 16B input buffer and 32B output buffer for each PE. Each NMP instruction necessitates the bank-group level FIFO instruction queue to determine whether it pertains to itself or a bank-level NMP. This determination is facilitated by a 2-bit flag `nmp_level` $= 10$. If the instruction belongs to the bank-group itself, it is decoded and directly accessed; otherwise, it is routed to the bank-level NMP. The input buffer is responsible for receiving data from the memory banks, while the output buffer transmits data to the near-rank NMP.

Bank-level NMP. Figure 5 (d) illustrates the near-bank NMP architecture. The FIFO instruction queue is designed to receive and store the C/A signals. Subsequently, the instruction decoder parses these instructions, extracting the memory address. In response, the memory arrays' row decoder will read the data located at the specified address into the row buffer. Then, the column decoder of memory arrays will read the corresponding data out. For sparse matrix multiplication, a 32-bit floating-point multiplier near the memory bank is incorporated. This multiplier efficiently accesses the row buffer with lower latency. Additionally, each near-bank NMP exclusively processes data associated with its designated dimension shards.

Softmax PE. Within the rank-level NMP, a dedicated softmax PE is incorporated to execute the row-wise softmax operation $softmax(s_i) = \frac{e^{s_i - s_{max}}}{\sum_{c=1}^{n} e^{s_c - s_{max}}}$, as illustrated in Figure 5 (e). The softmax unit adheres to the design principles of A^3 [12], with a distinctive partitioning of the exponential lookup table into upper and lower halves to manage the size of the lookup table more effectively.

C. NMP Instruction and C/A Bandwidth

Similar to RecNMP [16] and ReCross [22], SADIMM employs instruction compression, condensing signals, tags, and relevant details into an 82-bit instruction. Each component is delineated below. As shown in Figure 6, the 1-bit `D_mode` indicates the memory working on DRAM mode or NMP mode. The 2-bit `nmp_level` field determines whether an instruction is executed by the host, near-rank NMP, near bank-group NMP, or near-bank NMP. This classification is generated by the host-side memory controller based on operation types. The 3-bit `op_redu` field designates a reduction operation, encompassing summation and vectors' aggregation. The 3-bit DDR CMD field signifies DDR commands (ACT, RD, and PRE). The 34-bit `addr` field denotes the physical address of the target matrices' row vectors. The 3-bit `row_size` field indicates the row vector size and DRAM reads per row vector. The 32-bit `mat_mul` field represents the floating-point data for matrix multiplication. The `redu_tag` identifies if row vectors belonging to the same reduction operation are identically set, ensuring aggregation of only relevant data. The `batch_end` indicates the batch's end, with the last instruction setting it to 1, signaling that reduced results can be transferred to the host. For main memory consistency, all data sent to the host is written back to memory banks. The left 2-bits are reserved for the future use.

To alleviate potential PE stalling caused by delayed C/A signals, our objective is to establish an optimal scenario wherein an NMP instruction is transferred before its predecessor is completed. This approach synchronizes the instruction transfer time with the access time of the matrices' vectors, which is directly proportional to the vector length. To achieve above goals, akin to the strategy employed in TRiM [28], we implement a two-stage instruction transfer technique. This method utilizes both the DQ and C/A pins to transfer NMP instructions from the memory controller to the DRAM buffer chip. During matrix multiplication, where matrices' vectors are stored in PEs without occupying the DQ bus until the batch's completion, the idle DQ path can be repurposed for NMP instruction transfer.

V. SADIMM Dataflow

A. Dimension-based Data Sharding

As illustrated at the top of Figure 7, we implement a dimension-based partition for matrices W_Q, W_K, Q, K, and S, while employing a token-based partition for the remaining matrices (V and Z). We shard matrix S dimension-wise to utilize the tree-structure of DIMM to perform reductions efficiently. To ensure the correct calculation of SpMM, the matrix V must be sharded token-wise. Figure 7 adopts a specific configuration with $L = N = D = 3$, where L denoting the sequence length, N representing the number of banks, and D signifying the model dimension. We extract each dimension of all tokens to form a dimension-based vector, resulting in D vectors with L elements in each vector. The mapping strategy for the dimension-based partition is depicted in ①. We store the same dimension of W_Q and W_K in the same bank, while storing W_V in all banks. Our dimension-based sharding incurs the same intra-layer access overhead as token-based sharding, since there is no distinction between row-wise and column-wise access when processing the entire matrix. To minimize cross-layer transmissions and maximize cross-layer data reuse, we store the same part (here is dimensions) of all layers in the same bank, following a similar approach (here is tokens) to TransPIM [43].

This approach provides notable advantages, primarily as each memory bank is dedicated to a specific dimension for all tokens. In the scenario of sparse attention pruning for a weakly connected token, dimension-based sharding ensures the simultaneous pruning of all dimensions associated with weakly connected tokens across all banks. As a result, dimension-based sharding facilitates load balancing even when a considerable number of weakly connected tokens are pruned. Furthermore, this technique leverages data reuse across different layers

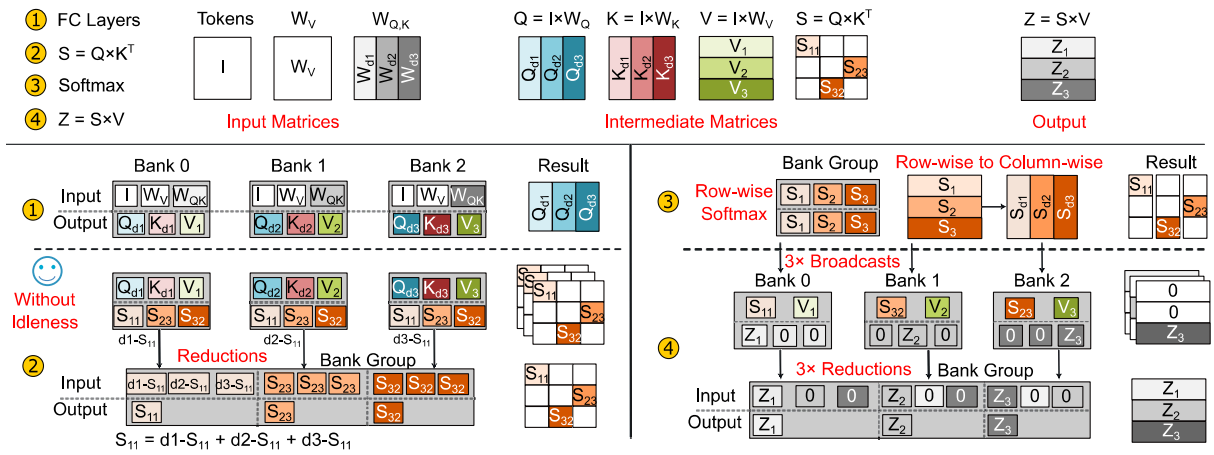

Fig. 7. The dimension-based dataflow of SADIMM

by consolidating computations of token dimensions in the same memory location. To support the dimension-based data partitioning, we write a scheduling program for the dataflow in Figure-6. The scheduling program will be loaded to the memory controller when DRAMs are in NMP mode.

B. Dimension-based Dataflow

Full-connection layer. Within the full connected layers, the input matrix I in bank$_i$ undergoes GEMM operations with the dimension shards of weight matrices W_Q and W_K. Specifically, $Q^{di} = I \times W_Q^{di}$ and $K^{di} = I \times W_K^{di}$ are performed, generating distinct dimensions of Q and K in each bank. For the generation of matrix V, the token-based dividing approach is adopted, where bank$_i$ executes $V_i = I_i \times W_V$, resulting in $\frac{L}{N} \times D$ elements per bank. Notably, the calculation in each bank-level PE for the full connected layers exclusively involves local access to the nearest bank, effectively eliminating potential PE stalls arising from data dependencies.

SDDMM between matrices Q and K. As depicted in ②, the output Q^{di} and K^{di} from the FC layer will serve as the input of SDDMM operation in bank$_i$. Subsequently, bank$_i$ performs the same-dimension multiplication to generate the i^{th} dimension of matrix S without requiring cross-bank transmissions. In the example illustrated in Figure 7, the first bank initiates the generation of the first dimension/slice of matrix S. To manage the data dependencies associated with cross-dimension accumulation, we leverage the broadcast and reduction capabilities of the DIMM-based NMP platform, as detailed in Section II-B. All banks transmit their dimensions/slices of matrix S to the bank-group PE, which undertakes cross-dimension accumulation to yield matrix S. Although the accumulation is done in the bank-group, our method introduces only small memory overhead. Because near-bank PEs process tokens one by one. When the intermediate results of current token is transmitted to the bank-group, the intermediate results of next token can be stored in the same place to save space. Despite a substantial number of tokens being pruned, the dimension-based SDDMM method does not introduce a load imbalance problem, as the dimensions of the retained tokens are evenly distributed across the memory banks.

Softmax operation. Executing the softmax operation within the dimension-based dataflow is straightforward. As shown in ③, the bank-group or rank-level PEs aggregate all dimensions from memory banks to generate matrix S (ranging from S_0 to S_N). To save memory space, we apply the softmax operation to S_0 immediately after its generation. However, the row-wise softmax introduces token-based sharding of matrix S, necessitating a subsequent reslicing phase to transform matrix S into dimension-based sharding. Given the limited memory arrays in the bank-group, we perform these operations sequentially from S_0 to S_N.

SpMM between matrices S and V. Considering the absence of memory arrays and the presence of only buffers in the bank-group, the dimension-based slices of matrix S generated by the bank-group PEs need storage back to memory banks through broadcasting. We employ dimension-based sharding for matrix S, with each bank accommodating $\frac{D}{N} \times L$ elements (both zeros and non-zeros). The computation between dimension-based S and token-based V mirrors the SDDMM operation of $Q \times K^{\mathsf{T}}$. As illustrated in ④, the calculation of $S \times V$ concludes with bank-level multiplication and bankgroup-level accumulation.

End-to-end Transformer. SADIMM is switchable between two memory modes, conventional *Memory Mode* (M-mode) and memory *Computing Mode* (C-Mode). Collaborating with SADIMM, the host processor performs the execution of the end-to-end Transformer. Specifically, SADIMM takes charge of offloading the self-attention kernel, while the host manages the remaining operations. Our experiments compare the performance of sparse attention kernel only.

Advantage: Fewer Cross-bank Transfers. In comparison to token-based dataflow, the dimension-based dataflow also reduces on-chip transmission with minimal cross-bank broadcasting. Specifically, same-dimension multiplication exhibits no data dependency with the information stored in other banks, enabling independent work for all near-bank PEs. In contrast, the token-based dataflow shown in Figure 4 involves many cross-bank token transfers, resulting in significant conflicts in DQ and C/A paths [43]. Furthermore, we utilize broadcasts to fulfill cross-dimension data dependencies, aligning well with the DIMM hierarchy.

TABLE I
SADIMM CONFIGURATIONS

Host Processor	16 out-of-order cores, 2.5GHz, 32KB L1 cache, 256KB L2 cache, 16MB LLC cache
DRAM Module	DDR4-2400MHz 8GB, 64GB total size, 4× Channels, 2× DIMMs, 2× Ranks, 4KB row buffer size, FR-FCFS
DRAM Timing Parameters	tRCD=16, tCL=16, tRP=16, tRC=55, tRRD_S=4, tRRD_L=6, tFAW=26, tCCD_S=4, tCCD_L=6, tBL=4
NMP Config.	16KB input buffer, 32KB output buffer, 2 FP32 adder and 1 softmax unit per rank-NMP, 1 FP32 adder per BG-NMP, and 1 FP32 multiplier per bank-NMP
Energy and Latency Parameters	DRAM ACT energy=2nJ, RD/WR=4.2pJ/bit, Off-chip I/O=4pJ/bit, FP32 adder=0.9pJ/Op, FP32 mult=2.4pJ/Op

VI. EVALUATION

A. Experimental Setup

Simulation. Table I provides a concise overview of the fundamental configurations of SADIMM. To comprehensively assess SADIMM's performance, we employ both Ramulator [17] and ZSim [32], a cycle accurate simulator. To model host interactions for NMP instruction offloading, token and weighted matrix retrieval, and data broadcast to the NMP side, we enhance the trace-driven version of ZSim. Within Ramulator, we introduce the DIMM-SADIMM module to emulate the sparse attention operations offloaded to the memory side. Utilizing CACTI [26], we estimate cache area, energy consumption, and access latency. The bandwidth of each near-bank NMP is configured to 1GB/s with our experiments. The total bandwidth of SADIMM is up to 1TB/s. The Synopsys Design Compiler, operating under a 40nm technology library, assesses the area, energy consumption, and latency of the integrated specialized hardware units. CACTI-3DD [4] gauges the energy consumption of DRAM devices, while CACTI-IO [15] evaluates the energy consumption of off-chip I/O and DIMM-level considerations.

Workloads. We evaluate SADIMM's performance across various *natural-language processing* (NLP) tasks using the BERT-base (BERT), BART, and GPT-2-Small (GPT2) models. For dynamic sparse attention, we apply Sanger's quantize-and-pruning method [23] to all models. For NLP inference, we utilize nine datasets from the *General Language Understanding Evaluation* (GLUE) [36] collection, including cola, mnli, mrpc, qnli, qqp, rte, sst-2, stsb, and wnli. The *maximal sequence length* (MSL) for all GLUE datasets is below 384. Additionally, we assess models on MSL 512 *Stanford Question Answer Dataset* (SQuAD v1.2) [31], MSL 1K WikiText-2 [25], and MSL 2K IMDB [24] datasets. Throughout this study, the maintained data precision is Float32.

Methodology. In our benchmarking analysis, we compare SADIMM against five contemporary designs on standard platforms. (1) CPU baseline: AMD Ryzen Threadripper 3970X; (2) GPU baseline: NVIDIA RTX A6000 GPU, featuring 46GB memory, 300W TDP, CUDA v11.6, and PyTorch v2.0.0 [29]; (3) Sanger [23]: an ASIC-based accelerator utilizing reconfigurable systolic arrays for sparse attention; (4) TRiM [28]: a DIMM-based NMP accelerator designed for accelerating recommendation systems through near rank/bank-group/bank processing. We add the softmax time of SADIMM to TRiM for the lack of softmax unit; (5) HAIMA [9]: a hybrid NMP leveraging near-SRAM and near-DRAM hardware.

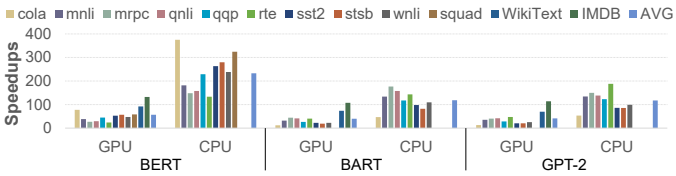

Fig. 8. The speedup improvement of SADIMM over GPU and CPU on BERT, BART, and GPT-2 benchmarks

Our pre-processing phases involve fine-tuning all models from pre-trained checkpoints using the corresponding training datasets to obtain weight matrices. These matrices are then pre-stored in memory banks with dimension-based sharding. Following this, we apply quantize-and-pruning sparse attention to obtain sparse mask matrices for all datasets, compressing them into *Compressed Sparse Row* (CSR) format. The sparse matrices are loaded and stored with CSR format. During the computation by the PEs, there is a coordinates alignment phase of the CSR format before the multiplication. For GLUE and SQuAD datasets, we maintain consistent learning rates and batch sizes as in Sanger [23]. Our code is adapted from the Sanger project on Github [23]. All models and datasets are sourced from the Hugging Face models library [37] and datasets library.

B. Comparison with CPUs and GPUs

We conduct experiments to evaluate the overall performance and energy efficiency of SADIMM in comparison to CPU and GPU platforms. In Figure 8, we present the achieved speedups by SADIMM relative to the CPU and GPU baselines. Some results are missing because the CPU was unable to obtain the results within the required time. Across BERT, BART, and GPT-2 datasets, SADIMM demonstrates average speedups of 47.3×, 35.2×, and 36.6× over the Nvidia RTX A6000 GPU, and 242.5×, 118.5×, and 117.7× over the AMD Ryzen CPU, respectively. Our experimental results show that SADIMM outperforms the CPU and GPU baselines in all benchmarks. This superiority arises from SADIMM's utilization of near-memory computation for sparse attention, resulting in a substantial reduction in off-chip memory access compared to the CPU and GPU baselines.

Figure 9 illustrates the energy efficiency gains achieved by SADIMM over the CPU and GPU baselines. For the BERT, BART, and GPT-2 models, SADIMM exhibited average energy efficiencies of 189.1×, 202.1×, and 191.3× compared to the GPU, and 348.6×, 470.9×, and 401.7× compared to the CPU platform. Across all datasets, SADIMM consistently outperformed CPU and GPU platforms in terms of energy efficiency. This heightened energy efficiency can be primarily attributed to the substantial reduction in data transfers between the host processor and off-chip main memory facilitated by SADIMM.

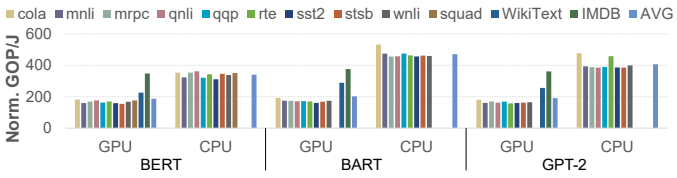

Fig. 9. The energy efficiency improvement of SADIMM over GPU and CPU on BERT, BART, and GPT-2 benchmarks

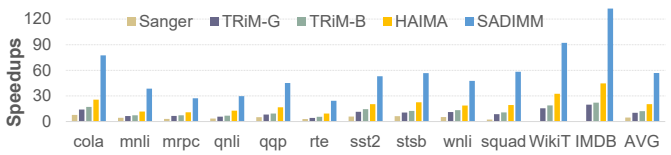

Fig. 10. Performance comparison with modern accelerators (normalized to GPU baseline)

C. Comparison with Other Accelerators

We conducted experiments to assess the overall performance of SADIMM in comparison to contemporary accelerators, including Sanger [23], TRiM [28], and HAIMA [9]. The performance results are illustrated in Figure 10, normalized to the GPU.

SADIMM vs. Sanger. Figure 10 presents a comparative analysis of the performance between the ASIC-based Sanger and SADIMM. Sanger introduces a prediction-based pruning method to minimize unnecessary calculations in the attention. These optimizations enable Sanger to achieve better performance, approximately 4.7× compared to GPU-based sparse attention solutions. Despite this, Sanger leaves certain challenges unresolved, which SADIMM addresses for further speedups. As an ASIC-based sparse attention accelerator, Sanger adheres to the conventional data access approach seen in other processor-centric architectures. Although Sanger rearranges non-zeros in the sparse matrix to enhance data reuse, the numerous intermediate matrices generated in the remaining calculations necessitate extensive off-chip transfers from DRAM to on-chip PEs. This off-chip transfer overhead significantly impacts Sanger's efficiency. In contrast, SADIMM's superior performance can be attributed to two key factors. First, SADIMM is a memory-based accelerator that can significantly reduce off-chip memory access. Second, SADIMM employs dimension-based data partitioning, allowing sparse matrices to be uniformly distributed by dimension in the memory bank. Therefore, SADIMM primarily involves random accesses within the bank, which has a higher bandwidth than Sanger's random accesses throughout the whole memory spaces.

SADIMM vs. TRiM. SADIMM demonstrates an average performance improvement of 5.6× and 4.7× compared to TRiM-G and TRiM-B, respectively. In contrast to GPUs, TRiM achieves an 11.3× performance boost by leveraging a memory-based architecture to mitigate off-chip transfers and introducing hot-entry replication to address load imbalance issues. While TRiM's homogeneous architecture proves effective for *gather and reduction* (GnR) operations in recommendation systems, it exhibits sub-optimal performance in supporting more intricate sparse attention tasks. Moreover, sparse attention, characterized by randomness and dynamics, poses challenges in identifying hot entries. SADIMM surpasses TRiM in performance, benefiting from two key aspects. First, SADIMM employs heterogeneous integration, providing dedicated logic units at different NMP levels for distinct operations. This strategy allocates appropriate hardware resources based on the bandwidth requirements of each operation, thereby enhancing overall bandwidth utilization. Second, SADIMM adopts dimension-based data partitioning, achieving balanced loads without the need for hot entry information and consequently reducing significant random scheduling overheads.

SADIMM vs. HAIMA. HAIMA employs two innovations to surpass GPU performance: 1) a hybrid NMP architecture, utilizing DRAM-based NMP for large-scale matrix multiplication and SRAM-based NMP as a filter between DRAM-based NMP and the host; 2) a parallel dataflow that accommodates varying parallelism and resource requirements among different layers. The above optimizations enable HAIMA to achieve a 20.5× performance improvement over GPUs. However, two challenges cause HAIMA's suboptimal performance: 1) Assuming equal access demands for all operations in the attention mechanism, leading to the allocation of the same hardware resources to diverse operations and resulting in wasted bandwidth; 2) While HAIMA's dataflow reduces cross-layer transfers, it introduces load imbalance when supporting sparse attention. In contrast, SADIMM introduces two optimizations for further performance enhancement. First, SADIMM employs heterogeneous integration, providing dedicated logic units at different NMP levels for distinct operations, allocating hardware resources based on the bandwidth requirements of each operation to enhance overall bandwidth utilization. Second, SADIMM adopts dimension-based data partitioning, achieving load balance without relying on hot entry information and thereby reducing significant random scheduling overheads.

D. Hardware and Software Efficiency

SADIMM constitutes a hardware-software co-designed sparse attention accelerator, where the interplay between hardware and software design is pivotal. In this section, experiments are conducted to compare SADIMM with two sister systems, aiming to assess the individual contributions of hardware and software optimizations. To gauge the hardware efficiency of SADIMM, Sister#1 is configured using our dimension-based dataflow on the NMP-based TRiM-B platform. Conversely, to assess the software efficiency of SADIMM, Sister#2 is configured using the token-based dataflow tailored for SADIMM. Figure 11 presents the average performance of each model on each dataset, with all performance metrics normalized to the speedups of the GPU baseline.

Hardware efficiency. Figure 11 illustrates that Sister#1 achieves average speedups of 26.1× and 2.14× compared to the GPU baseline and TRiM-B, respectively. The enhancement over the GPU baseline is attributed to the reduction in processor-memory transmissions through the utilization of NMP architecture. The improvement over TRiM-B is primarily

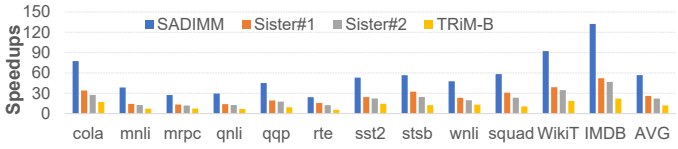

Fig. 11. Performance comparison with two sister platforms and TRiM-B (normalized to GPU baseline)

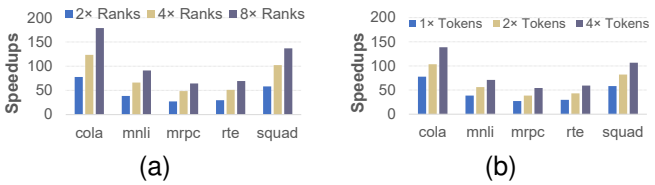

Fig. 12. (a) The speedups with various number of ranks, and (b) The speedups with various number of tokens (normalized to GPU baseline)

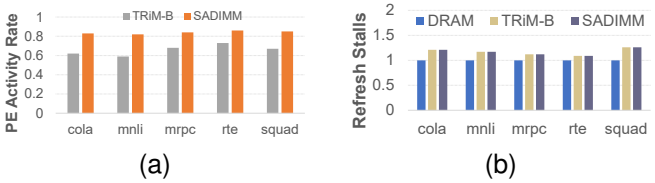

Fig. 13. (a) PE activity rate comparison with baseline platforms, and (b) Refresh stalls comparison with baseline platform (normalized to DRAM)

due to the dimension-based dataflow, which achieves load balance and reduces numerous random scheduling overheads. However, the Sister#1 system only attains 46% of the performance compared to SADIMM. This is because Sister#1 integrates the same logical units to support diverse operations in sparse attention, leading to three challenges, as analyzed in Section III-A. In contrast, SADIMM adopts heterogeneous integration, allocating different logical units to support diverse operations. This reduces resource contention between operations and enhances bandwidth utilization, resulting in higher speedups over Sister#1.

Software efficiency. The Sister#2 platform achieves speedups of 22.3× and 1.84× compared to the GPU baseline and TRiM-B, respectively. The improvement over the TRiM-B platform can be attributed to heterogeneous integration, achieving a better match between hardware resources and the access requirements of operations. However, the Sister#2 platform only achieves 39% of the performance compared to SADIMM. This is because the Sister#2 platform adopts a token-based dataflow, leading to a severe load imbalance problem. To address this imbalance, Sister#2 utilizes the on-chip scheduler to realign data, introducing significant on-chip scheduling overhead and many cross-bank transmissions. In contrast, SADIMM employs dimension-based dataflow, achieving balanced loads naturally and greatly reducing runtime scheduling overhead.

E. Scalability

Numbers of Ranks. Figure 12 (a) illustrates the performance of SADIMM varies with the number of ranks. Although the rank-level bandwidth increases with the number of ranks, the DIMM-level bandwidth does not increase linearly due to shared data cables between ranks. Nevertheless, SADIMM exhibits improved performance with an increasing number of ranks. For instance, there is a 1.69× performance improvement for four ranks compared to two ranks. This enhancement is attributed to the effective utilization of rank/bank-group/bank-level bandwidth. By integrating logical units in each rank/bank-group/bank, SADIMM leverages higher near-memory bandwidth as the number of ranks increases. This highlights the superior scalability of SADIMM with memory bandwidth compared to the GPU baseline.

Numbers of Tokens. Facilitating the processing of long sequences is pivotal for sparse attention accelerators. To assess the sequence scalability of SADIMM, we extend our evaluation by generating longer sequence datasets through repetition, doubling and quadrupling the original sequence. Figure 12 (b) depicts the speedups achieved by SADIMM compared to the GPU baseline for varying sequence lengths. Notably, as the sequence length increases, SADIMM demonstrates higher speedups, showcasing its superior scalability in handling long sequences compared to the GPU baseline. This scalability is attributed to SADIMM's utilization of NMP hardware, which significantly mitigates off-chip random access, which is a primary bottleneck for the GPU baseline that grows quadratically with sequence length [19].

F. Discussion

PE Activity Rate. To reveal the hardware utilization efficiency of SADIMM, Figure 13 (a) presents the PE activity rate of SADIMM and TRiM-B. Compared with TRiM-B's 60% PE activity rate, SADIMM has a higher PE activity rate of about 80%. The improvement of PE utilization of SADIMM can be contributed to our dimension-based dataflow, which can significantly reduce cross-bank transmissions. Fewer cross-bank transfers will reduce PE access overhead and increase PE utilization.

Analysis of Refresh Stalls. Figure 13 (b) evaluates the refresh stalls of DRAM, TRiM-B, and SADIMM. We define refresh stalls as PE stalls caused by periodic memory refresh. Compared to DRAM, both TRiM-B and SADIMM have a higher refresh stalls, due to a higher memory utilization reducing the time-overlap opportunity for many idle memory banks. Our results also show that SADIMM does not result in increased refresh stalls compared to TRiM-B even with higher PE utilization. The common used time-overlap method involves refreshing idle banks while other active banks are engaged in computation. However, in the baseline platform, while some PEs may be idle, the memory banks themselves are always busy, as shown in Figure 4. This is due to cross-bank data access, where one bank may be serving data to the PE of another bank. SADIMM's goal is to reduce near-bank PE idle time, which does not change the idleness of memory arrays. Therefore, the methods in SADIMM do not directly correlate with an increase in refresh stalls.

G. Energy and Area Breakdown

DIMM-based NMP accelerators integrate logic units directly into the DRAM chip, imposing additional area and

TABLE II
POWER AND AREA BREAKDOWN OF SADIMM AND TRIM

Name	Component	PE type	Area (mm^2)	Power (mW)
SADIMM	Rank PEs	adder&soft.	0.73	82
	BG PEs	adder	0.39	37
	Banks PEs	multiplier	2.29	216
TRiM-B	Rank PEs	MAU	0.36	54
	BG/bank PEs	MAU	11.5	1.3K

power overhead on DRAM. We present the area and power consumption details of SADIMM in Table II, which contains 2× DIMMs, 2× ranks, 8× DRAM chips, 8× bank-groups, and 4× banks. In a 40nm technology context, SADIMM exhibits a minimal area overhead of 3.42 mm^2, a considerable reduction compared to the 11.86 mm^2 area of TRiM-B. Current NMP platforms integrate powerful PEs to support diverse operations. In contrast, SADIMM integrates small area adder or multiplier to support some specific arithmetic operations. Furthermore, the power consumption of SADIMM is measured at 335 mW, significantly lower than the total power of TRiM-B, which stands at 1.3W.

VII. RELATED WORK

Accelerating Sparse Attention. Research aimed at accelerating sparse attention initially flourished on GPU platforms due to their enhanced computing parallelism [3], [6]–[8], [35]. GPU-based solutions [3], [6], [7] focus reducing memory requirements during attention computation. However, the irregular distribution of non-zeros in sparse attention introduces a substantial random access overhead. To address these challenges, hardware-software co-design approaches, such as FPGA [2], [41] and ASIC [12], [23], [30] platforms, have emerged. These platforms aim to overcome the inherent latency-bound inefficiencies associated with limited sparse data reuse. However, the performance of these processor-centric solutions is constrained as the sequence length expands, with off-chip memory access becoming a bottleneck [38].

NMP and PIM Platforms. NMP platforms can reduce off-chip transfers and concurrently enhances the bandwidth of in-memory logic units. Demonstrating prowess in improving performance for memory-intensive applications, including neural networks [11], [21], [39], graph processing [5], [34], [40], and attention mechanisms [9], [20], [43]. The above NMP-based solutions exhibit superior bandwidth utilization. However, they grapple with challenges such as PE idleness and on-chip scheduling overhead, arising from homogeneous integration and irregular pruning. This work aims to address these challenges by proposing heterogeneous integration and a dimension-based dataflow. In addition to NMP-based solutions, there are also PIM-based attention mechanism accelerators [19], [42], which reduce off-chip random access through the computational capabilities of memory arrays.

VIII. CONCLUSION

We introduce SADIMM, a novel co-design sparse attention accelerator that utilizes NMP platforms. First, we develop heterogeneous NMP architecture, integrating different logical units in the memory hierarchy to support various operations with high hardware efficiency. Second, we introduce dimension-based dataflow, dividing and storing input sequences by model dimensions to avoid PE idleness. The experimental outcomes demonstrate that SADIMM exhibits remarkable performance and energy efficiency compared to state-of-the-art sparse attention accelerators.

REFERENCES

[1] A. Arnab, M. Dehghani, G. Heigold, C. Sun, M. Lučić, and C. Schmid, "ViViT: A Video Vision Transformer," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2021, pp. 6836–6846.

[2] Z. Bai, P. Dangi, H. Li, and T. Mitra, "SWAT: Scalable and Efficient Window Attention-based Transformers Acceleration on FPGAs," 2024. [Online]. Available: https://arxiv.org/abs/2405.17025

[3] I. Beltagy, M. E. Peters, and A. Cohan, "Longformer: The Long-Document Transformer," *CoRR*, vol. abs/2004.05150, 2020.

[4] K. Chen, S. Li, N. Muralimanohar, J. H. Ahn, J. B. Brockman, and N. P. Jouppi, "CACTI-3DD: Architecture-level modeling for 3D die-stacked DRAM main memory," in *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2012, pp. 33–38.

[5] G. Dai, Z. Zhu, T. Fu, C. Wei, B. Wang, X. Li, Y. Xie, H. Yang, and Y. Wang, "DIMMining: Pruning-Efficient and Parallel Graph Mining on near-Memory-Computing," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, 2022, p. 130–145.

[6] T. Dao, "FlashAttention-2: Faster attention with better parallelism and work partitioning," 2023.

[7] T. Dao, D. Fu, S. Ermon, A. Rudra, and C. Ré, "FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness," in *Advances in Neural Information Processing Systems*, vol. 35, 2022, pp. 16 344–16 359.

[8] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," *CoRR*, vol. abs/1810.04805, 2018.

[9] Y. Ding, C. Liu, M. Duan, W. Chang, K. Li, and K. Li, "HAIMA: A Hybrid SRAM and DRAM Accelerator-in-Memory Architecture for Transformer," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*, 2023, pp. 1–6.

[10] Y. Fang, B. Liao, X. Wang, J. Fang, J. Qi, R. Wu, J. Niu, and W. Liu, "You Only Look at One Sequence: Rethinking Transformer in Vision through Object Detectio," *CoRR*, vol. abs/2106.00666, 2021.

[11] M. Gao, J. Pu, X. Yang, M. Horowitz, and C. Kozyrakis, "TETRIS: Scalable and Efficient Neural Network Acceleration with 3D Memory," in *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, Apr 2017.

[12] T. J. Ham, S. J. Jung, S. Kim, Y. H. Oh, Y. Park, Y. Song, J.-H. Park, S. Lee, K. Park, J. W. Lee, and D.-K. Jeong, "A^3: Accelerating attention mechanisms in neural networks with approximation," in *Proceedings of 2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2020, pp. 328–341.

[13] M. Imani, S. Gupta, Y. Kim, and T. Rosing, "FloatPIM: In-Memory Acceleration of Deep Neural Network Training with High Precision," in *Proceedings of the 46th International Symposium on Computer Architecture*, 2019, p. 802–815.

[14] JEDEC, "DDR5 SDRAM STANDARD," 2020. [Online]. Available: https://www.jedec.org/standards-documents/docs/jesd79-5b

[15] N. P. Jouppi, A. B. Kahng, N. Muralimanohar, and V. Srinivas, "CACTI-IO: CACTI with off-Chip Power-Area-Timing Models," in *Proceedings of the International Conference on Computer-Aided Design*. Association for Computing Machinery, 2012, p. 294–301.

[16] L. Ke, U. Gupta, B. Y. Cho, D. Brooks, V. Chandra, U. Diril, A. Firoozshahian, K. Hazelwood, B. Jia, H.-H. S. Lee, M. Li, B. Maher, D. Mudigere, M. Naumov, M. Schatz, M. Smelyanskiy, X. Wang, B. Reagen, C.-J. Wu, M. Hempstead, and X. Zhang, "RecNMP: Accelerating Personalized Recommendation with Near-Memory Processing," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020, pp. 790–803.

[17] Y. Kim, W. Yang, and O. Mutlu, "Ramulator: A fast and extensible DRAM simulator," *IEEE Computer Architecture Letters*, vol. 15, no. 1, pp. 45–49, 2016.

[18] H. Li, H. Jin, L. Zheng, Y. Huang, X. Liao, Z. Duan, D. Chen, and C. Gui, "ReSMA: accelerating approximate string matching using ReRAM-based content addressable memory," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, p. 991–996.

[19] H. Li, H. Jin, L. Zheng, X. Liao, Y. Huang, C. Liu, J. Xu, Z. Duan, D. Chen, and C. Gui, "CPSAA: Accelerating Sparse Attention Using Crossbar-Based Processing-In-Memory Architecture," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2023.

[20] H. Li, Z. Li, Z. Bai, and T. Mitra, "ASADI: Accelerating Sparse Attention Using Diagonal-based In-Situ Computing," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2024, pp. 774–787.

[21] C. Liu, H. Liu, H. Jin, X. Liao, Y. Zhang, Z. Duan, J. Xu, and H. Li, "ReGNN: a ReRAM-based heterogeneous architecture for general graph neural networks," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, p. 469–474.

[22] H. Liu, L. Zheng, Y. Huang, C. Liu, X. Ye, J. Yuan, X. Liao, H. Jin, and J. Xue, "Accelerating Personalized Recommendation with Cross-Level Near-Memory Processing," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, 2023.

[23] L. Lu, Y. Jin, H. Bi, Z. Luo, P. Li, T. Wang, and Y. Liang, "Sanger: A Co-Design Framework for Enabling Sparse Attention Using Reconfigurable Architecture," in *Proceedings of 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, p. 977–991.

[24] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, June 2011, pp. 142–150.

[25] S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer sentinel mixture models," *arXiv preprint arXiv:1609.07843*, 2016.

[26] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, "CACTI 6.0: A tool to model large caches," *HP laboratories*, vol. 27, p. 28, 2009.

[27] O. Mutlu, S. Ghose, J. Gómez-Luna, and R. Ausavarungnirun, "A modern primer on processing in memory," in *Emerging Computing: From Devices to Systems: Looking Beyond Moore and Von Neumann*. Springer, 2022, pp. 171–243.

[28] J. Park, B. Kim, S. Yun, E. Lee, M. Rhu, and J. H. Ahn, "TRiM: Enhancing Processor-Memory Interfaces with Scalable Tensor Reduction in Memory," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, p. 268–281.

[29] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32, 2019.

[30] Z. Qu, L. Liu, F. Tu, Z. Chen, Y. Ding, and Y. Xie, "DOTA: Detect and Omit Weak Attentions for Scalable Transformer Acceleration," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2022, p. 14–26.

[31] P. Rajpurkar, R. Jia, and P. Liang, "Know what you don't know: Unanswerable questions for SQuAD," *arXiv preprint arXiv:1806.03822*, 2018.

[32] D. Sanchez and C. Kozyrakis, "ZSim: Fast and accurate microarchitectural simulation of thousand-core systems," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*. Association for Computing Machinery, 2013, p. 475–486.

[33] W. Sun, Z. Li, S. Yin, S. Wei, and L. Liu, "ABC-DIMM: Alleviating the Bottleneck of Communication in DIMM-based Near-Memory Processing with Inter-DIMM Broadcast," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, 2021, pp. 237–250.

[34] N. Talati, H. Ye, Y. Yang, L. Belayneh, K.-Y. Chen, D. Blaauw, T. Mudge, and R. Dreslinski, "NDMiner: Accelerating Graph Pattern Mining Using near Data Processing," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, 2022, p. 146–159.

[35] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[36] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman, "GLUE: A multi-task benchmark and analysis platform for natural language understanding," *arXiv preprint arXiv:1804.07461*, 2018.

[37] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. Le Scao, S. Gugger, M. Drame, Q. Lhoest, and A. Rush, "Transformers: State-of-the-Art Natural Language Processing," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, Oct. 2020, pp. 38–45.

[38] X. Xie, Z. Liang, P. Gu, A. Basak, L. Deng, L. Liang, X. Hu, and Y. Xie, "SpaceA: Sparse Matrix Vector Multiplication on Processing-in-Memory Accelerator," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2021, pp. 570–583.

[39] J. Xu, H. Liu, Z. Duan, X. Liao, H. Jin, X. Yang, H. Li, C. Liu, F. Mao, and Y. Zhang, "ReHarvest: An ADC Resource-Harvesting Crossbar Architecture for ReRAM-Based DNN Accelerators," *ACM Trans. Archit. Code Optim.*, vol. 21, no. 3, Sep. 2024.

[40] M. Zhang, Y. Zhuo, C. Wang, M. Gao, Y. Wu, K. Chen, C. Kozyrakis, and X. Qian, "GraphP: Reducing Communication for PIM-Based Graph Processing with Efficient Data Partition," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Feb 2018.

[41] X. Zhang, Y. Wu, P. Zhou, X. Tang, and J. Hu, "Algorithm-Hardware Co-Design of Attention Mechanism on FPGA Devices," *ACM Trans. Embed. Comput. Syst.*, vol. 20, no. 5s, pp. 1–24, 2021.

[42] Q. Zheng, S. Li, Y. Wang, Z. Li, Y. Chen, and H. H. Li, "Accelerating Sparse Attention with a Reconfigurable Non-volatile Processing-In-Memory Architecture," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*, 2023, pp. 1–6.

[43] M. Zhou, W. Xu, J. Kang, and T. Rosing, "TransPIM: A Memory-based Acceleration via Software-Hardware Co-Design for Transformer," in *Proceedings of 2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2022, pp. 1071–1085.

Hui-Ze Li received his Ph.D. degree from the School of Computer Science and Technology, Huazhong University of Science and Technology (HUST), in 2022. He is now working as a Research Fellow in School of Computing, National University of Singapore (NUS). His current research interests include computer architecture, emerging non-volatile memory, machine learning accelerator, and processing in memory.

Dan Chen received the Ph.D. degree from the School of Computer Science and Technology, Huazhong University of Science and Technology (HUST), in 2024. He is now working as a Research Fellow in School of Computing, National University of Singapore (NUS). His research interests focus on processing-in-memory and graph neural network.

Tulika Mitra received a BE degree in computer science from Jadavpur University, Kolkata, India, in 1995, an ME degree in computer science from the Indian Institute of Science, Bengaluru, India, in 1997, and a Ph.D. degree from the State University of New York, Stony Brook, NY, USA, in 2000. She is currently Provost's Chair Professor of Computer Science at the School of Computing, National University of Singapore, Singapore. Her research interests include the design automation of embedded realtime systems with particular emphasis on software timing analysis/optimizations, application-specific processors, energy-efficient computing, and heterogeneous computing.