

MACAM: A Flexible Computing-in-Memory Accelerator for Sparse Matrix-Dense Vector Multiplication

Xiaoyu Zhang¹, Rui Liu^{1*}, Zerun Li¹, Yinhe Han¹, Xiaoming Chen^{1*}

¹Institute of Computing Technology, Chinese Academy of Sciences

*Corresponding authors (e-mail: chenxiaoming@ict.ac.cn, liurui@ict.ac.cn)

Abstract—Sparse Matrix-Dense Vector Multiplication (SpMV) is an important computational primitive which is bounded by memory bandwidth. Computing-in-memory (CIM) is regarded as an effective approach to reduce data movement. Due to the lack of flexibility in architectural design, current CIM-based SpMV accelerators struggle to simultaneously support high-parallelism computations and the storage of irregular sparse data. We propose a flexible CIM-based accelerator named MACAM for high-precision SpMV. Each array of MACAM can be configured into sparse or dense modes according to the local-sparsity of the sparse matrix. We propose a unified data layout approach that enables MACAM to meet the data storage requirements of different modes. We also propose a sparse storage format and a workload-balancing approach to further improve the performance of MACAM. Experiments show that MACAM achieves $167.26\times$ speedup and $286.04\times$ energy saving over the GPU baseline. MACAM also achieves $97.41\times$ and $6.56\times$ speedup and $213.65\times$ and $10.06\times$ energy saving compared with two state-of-the-art CIM-based SpMV accelerators.

Index Terms—Computing-in-memory, Sparse matrix-dense vector multiplication, Non-volatile memory

I. INTRODUCTION

Sparse matrix-dense vector multiplication (SpMV) is widely used in many fields such as sparse iterative linear solvers [1], [2], graph analytics [3], [4] and machine learning [5], [6]. Performing SpMV on real-world sparse matrices, which are large in size but have low density [7], will result in irregular memory accesses and low bandwidth utilization. The computing-in-memory (CIM) technique can reduce data movement and alleviate the memory wall bottleneck. Many CIM accelerators have been proposed to accelerate SpMV [8]–[13]. Depending on the hardware components used, CIM SpMV accelerators can be divided into two categories: dense approach and sparse approach based accelerators. Dense approach based accelerators only use multiply-accumulate (MAC) arrays. Sparse approach based accelerators are composed of both MAC and content-addressable memory (CAM) arrays.

The advantage of the dense approach is its high computational parallelism. When a matrix is mapped to a MAC array, the MAC array can complete the multiplication of the matrix with the dense vector in a highly parallel manner. Taking this advantage into account, some dense approach based accelerators [8], [9] have been proposed. However, when

there are no other non-zero elements near a certain non-zero element in the matrix, the situation becomes problematic. Due to severe resource wastage, such non-zero elements cannot be effectively mapped onto accelerators using the dense approach. Therefore, these non-zero elements need to be transferred to GPU for computation, significantly reducing the performance.

To map irregular sparse data in a sparse matrix, some sparse approach based accelerators [10]–[13] that use sparse storage formats have been proposed. Apart from using MAC arrays, these accelerators also use CAM arrays to store the row and column indexes of non-zero elements. CAM arrays, which can perform one search at a time, are used for searching for matched non-zero elements for computation. Using the sparse approach, data with different indexes stored in a MAC array cannot participate in the calculation at the same time, thereby reducing the computational parallelism. The additional search process further lowers the performance. It is difficult for the above two types of fixed architectures based accelerators to simultaneously achieve high computational parallelism and effectively store irregular sparse data, which poses new challenges for accelerating SpMV using the CIM technique.

The essential reason for the above challenge is that data in different regions of a sparse matrix have different local sparsity [14]. Since the architecture is fixed, current CIM-based accelerators have to choose between using a dense or sparse approach and cannot exploit the local sparsity of the sparse matrix. Considering that different matrices have different data distribution patterns and local sparsity, directly using a heterogeneous architecture composed of two types of fixed-size hardware components to accelerate SpMV on different matrices would be inefficient. **Therefore, a flexible accelerator that can support different storage and computing modes is necessary for accelerating SpMV.**

Flexible CIM accelerators have been proposed to accelerate neural network inference [15], [16]. However, current flexible CIM accelerators have not placed emphasis on accelerating SpMV. The lack of unified data layout and working approach prevents them from leveraging flexibility in accelerating SpMV. Considering both the dense and sparse approaches and the advantages of flexible accelerators, we propose a flexible CIM accelerator named MACAM for high-precision SpMV. The contributions of this paper are as follows.

- We design a flexible architecture for floating-point SpMV,

This work was supported in part by National Natural Science Foundation of China under Grant 62488101, Grant 62495104, and Grant 62025404, and in part by Youth Innovation Promotion Association CAS.

thereby applying the advantages of flexible CIM accelerators to the field of sparse function acceleration for the first time. Each array of MACAM can be configured into sparse mode or dense mode, allowing it to flexibly adapt to the storage and computational needs of different regions with different sparsity.

- We propose a unified data layout approach that not only meets the data storage requirements for both sparse and dense modes, but also enables the multifunctional arrays of MACAM to flexibly switch between different modes.
- We further propose a modified sparse storage format and a workload balancing approach for the flexible architecture to boost the performance of MACAM.
- Experimental results show that MACAM achieves $167.26\times$ speedup and $286.04\times$ energy saving over the GPU baseline. When compared with two state-of-the-art CIM SpMV accelerators [8], [11], MACAM achieves $97.41\times$ and $6.56\times$ speedup and $213.65\times$ and $10.06\times$ energy saving, respectively.

II. BACKGROUND

A. CIM-based Accelerators for SpMV

Resistive random-access memory (ReRAM) is widely used in CIM accelerators. A ReRAM-based MAC array is composed of a crossbar and peripheral circuits such as digital-to-analog converters (DACs) and analog-to-digital converters (ADCs). The MAC array is used to perform MAC operations between stored dense matrix data and input dense vector data. Some researchers have proposed using such a dense approach to accelerate SpMV [8], [9]. However, such a dense approach is unable to fully map irregular sparse matrix data.

A CAM array is used to compare an input key with stored values in parallel and output whether each value matches the input key. In previous sparse approach based accelerators [10]–[13], CAM arrays are used to store indexes of sparse matrix data represented in the sparse storage format. The accelerator first searches within the CAM arrays for matching indexes, and then based on the search results, determines which rows in the MAC array would be involved in the computation. During this process, only a subset of rows in the MAC array can participate in the computation. Such a sparse approach introduces additional search overhead and reduces the computational parallelism of MAC arrays.

Apart from differing in computational approaches, current CIM SpMV accelerators also target different applications. Some SpMV accelerators [9], [17], [18] are designed for graph algorithms, where the data involved is low-precision integers. Other accelerators [8], [11], [13] are designed to accelerate the SPMV kernel in scientific computing, where the data involved is typically high-precision ((32-bit or 64-bit)) floating-points. These accelerators not only need to support SpMV but also need to support floating-point MACs. MACAM is designed for accelerating high-precision SpMV kernels in scientific computing, such as iterative solution of linear equations, and therefore, it also needs to support floating-point calculations.

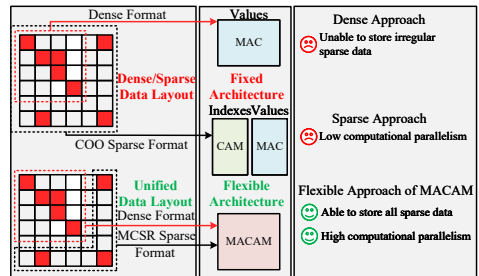


Fig. 1: Motivation of MACAM.

B. Sparse Storage Format

Sparse matrix data is typically stored using sparse storage formats, such as Coordinate List (COO) format, Compressed Sparse Row (CSR) format, and Compressed Sparse Column (CSC) format, etc. The COO format consists of three parts: row indexes, column indexes and values. The row and column indexes can be stored in a CAM array, enabling rapid searching for non-zero elements. The COO format is widely used in current CIM-based SpMV accelerators [10]–[13].

The CSR and the CSC format are quite similar. Taking the CSR format as an example, it consists of three parts: row offsets, column indexes and values. The row offsets are used to record the start and end positions of non-zero elements in each row. To obtain the number of elements in each row and determine the range of data for each row, it is necessary to obtain two adjacent row offsets and subtract them. Additional subtractions lead to extra data reading and calculation that will disrupt the rapid parallel search capability of CAM arrays. Consequently, it is challenging to effectively integrate CSR and CSC formats with CAM arrays, making it difficult for current CIM accelerators to leverage the advantages of these compressed sparse storage formats.

C. Motivation

As shown in Fig. 1, current CIM-based SpMV accelerators with a fixed architecture can only utilize either the sparse or dense approach to accelerate SpMV. Regardless of whether using a dense or sparse approach, current fixed-architecture CIM accelerators struggle to simultaneously achieve high computational parallelism and store the irregular sparse data. Due to the fixed architecture and differences in data layout as well as hardware components, the two different approaches cannot be simply and directly combined. We propose a flexible accelerator, MACAM, to overcome this limitation. By exploiting the local sparsity of sparse matrices and flexibly configuring hardware resources, MACAM can accelerate SpMV in a more flexible and efficient manner.

III. ARCHITECTURE OF MACAM

A. Overall Architecture and Multifunctional Array Design

MACAM supports two working modes: dense and sparse mode. In dense mode, the arrays are configured as MAC arrays. In sparse mode, the arrays are configured as both CAM and MAC arrays. Overall, MACAM is comprised of 10 PEs, where each PE consists of multiple arrays and some shared hardware components. Fig. 2 shows the architecture of a PE in MACAM. The shared hardware components in each PE are

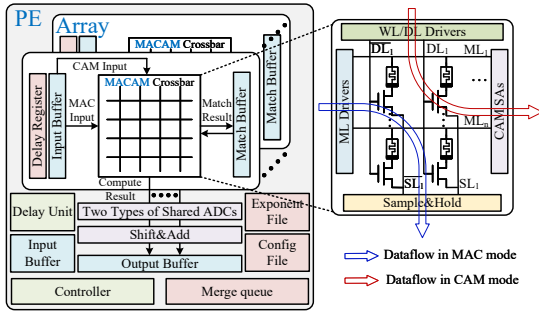


Fig. 2: Overall architecture of MACAM.

reused by the arrays within the PE. Computing components include input and output buffers, shared ADCs, and shift-and-add units, which assist the arrays in completing MAC calculations. Considering the different row utilization in sparse and dense modes, two types of ADCs with different precisions are integrated. MACAM supports SpMV calculations on IEEE-754 double-precision floating-point data [19] using a delay-based approach. Delay units and exponent files are used to support high-precision SpMV calculations. The merge queue is used to merge intermediate results and the configuration file is used to store configuration information. The controller is responsible for controlling the entire computation process.

PE. Each PE in MACAM contains 64 arrays, and each array is composed of a single-level cell based ReRAM multifunctional crossbar and peripheral circuits. The crossbar whose size is 128×128 is divided into four regions during computation. Each region is referred to as a sub-array. MACAM enables flexible configuration at the sub-array level. The delay register is used to store delay values and works together with delay units. The input buffer is used to store input data and query keys. The match buffer is used to store matching results.

Array. Each array in MACAM can be configured as either a MAC or CAM array, which is the source of its flexibility and also the origin of its name. Fig. 2 shows the different data flows within the multifunctional array configured in CAM or MAC mode. When an array is configured as a CAM array, the adjacent ReRAM devices on the same row will form a CAM cell. The search key will be input through the data lines, and the match result for each row will be transmitted via the match lines and converted into a digital signal by the CAM sense amplifiers. When an array is configured as a MAC array, each ReRAM cell within it will store a bit of the matrix data. Vector data will be input through the match lines and then multiplied with the matrix data stored in the ReRAM-based crossbar. The output will be converted into digital signals via one of two ADCs, selected by multiplexers controlled by mode configuration information. The multifunctional array provides the hardware foundation for MACAM to support high-parallelism computation in dense mode and irregular sparse data storage in sparse mode.

B. Data Layout and MCSR Format

The data layouts of CAM arrays and MAC arrays are different, which hinders flexible conversion between them. To map adjacent rows and columns of a sparse matrix onto a MAC array, high-precision data stored in a MAC array needs to be

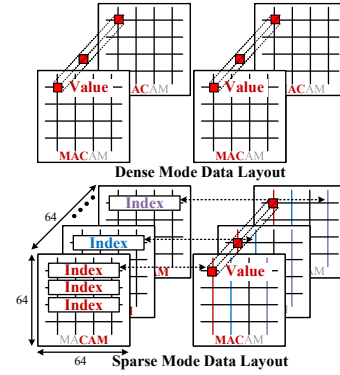


Fig. 3: Data layout of dense and sparse modes.

distributed across multiple arrays using a bit-sliced fashion. On the other hand, to enable parallel comparison of each bit between the query key and the stored data, a CAM array needs to store each bit of the data in the same row. To address this, a unified data layout approach is proposed and shown in Fig. 3.

Unified Data Layout. Regardless of whether it is in sparse mode or dense mode, the MAC arrays in MACAM store different bits of data from the sparse matrix using a bit-sliced approach. In sparse mode, data is stored in a sparse storage format, where the CAM arrays store the indexes of the data arranged in row-major order and the MAC arrays store the values of the data. Each bit of a single index is stored in the same row of a CAM array. The indexes stored in a CAM array correspond one-to-one with the values stored in a MAC array. For a PE, the data values corresponding to the indexes from the first CAM array are stored in the first column of a set of MAC arrays, the data values corresponding to the second CAM array are stored in the second column, and so on. To ensure this correspondence, the row and column dimensions, the bit-slice size of the sub-arrays are all set to the same value.

MCSR Format. To further save storage space, we propose a modified CSR (MCSR) storage format for the sparse mode. As shown in Fig. 4, the MCSR format modifies the row offsets in the CSR format to row counts. The row count values are stored in the registers corresponding to the starting position of the column indexes for each row, thereby serving the same indicative function as the row offset. The primary advantage of using row counts is that it avoids the need to subtract row offsets before each CAM search, which not only enhances performance but also eliminates the need to integrate additional subtractors near each array. Registers are required to store the row counts of the MCSR format. When an array is configured in CAM mode, the delay registers, which are used to assist MAC operations, are not in use. Therefore, these delay registers can be reused to store the row counts. The MCSR

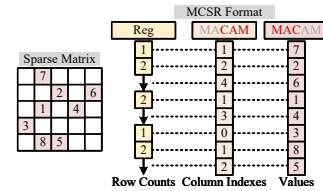


Fig. 4: MCSR format used in the sparse mode of MACAM.

format and the unified data layout enable MACAM to store all irregular sparse data within different sparse matrices.

C. Mode Configuration and Working Approach

The computation mode of MACAM hardware resources is configured based on the local sparsity of sparse matrices. As shown in Fig. 5, MACAM uses a one-time static data layout preprocessing algorithm to complete the configuration. The algorithm traverses the matrix, observing the local sparsity of each data block that matches the size of a sub-array, and skips completely blank areas. When the sparsity of a region exceeds a predefined threshold, the sub-arrays for that region will be configured in dense mode. Conversely, the sub-arrays for that region will be configured in sparse mode. In cases where the data within a sparse region is insufficient to fill a sub-array configured in sparse mode, data from other regions can be further deployed onto sub-arrays already configured in sparse mode. The threshold used during the traversal process is set to a theoretical value derived from calculating and comparing the computation times of sub-arrays deployed in dense mode versus sparse mode based on hardware parameters. Given the limited lifetime of ReRAM, MACAM is not well-suited for scenarios where matrices change frequently. Considering that, the target application of MACAM is sparse iterative solving, with relatively fixed matrices. As a static data layout algorithm, this preprocessing algorithm runs only once and will not significantly impact system performance, which will be shown in the application-level evaluation.

Workload Balance. Due to the different computation processes between dense and sparse mode, sub-arrays configured in these two modes operate at different speeds, leading to an imbalance in workload. To alleviate this issue, during resource allocation, MACAM combines and allocates sub-arrays configured in dense modes and sub-arrays configured in sparse modes across the array of the same PE. This approach prevents all sub-arrays configured in sparse mode from being allocated to a single PE, thereby balancing the workload and enhancing overall performance.

Working Approach. The working stages of MACAM are composed of writing, computing and merging stage. Fig. 6 shows the data flow and components used in the writing and computing stages for sparse and dense modes. In the writing stage of dense mode, MACAM writes matrix data with the dense data layout into arrays. For the writing stage in sparse mode, MACAM not only needs to write matrix data stored in MCSR format into arrays according to the sparse data layout, but also needs to write the row counts in MCSR

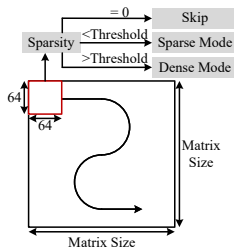


Fig. 5: Local sparsity aware mode configuration.

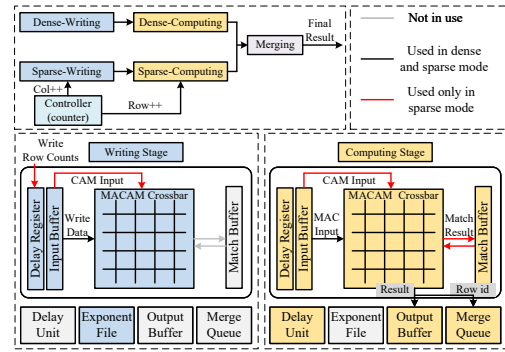


Fig. 6: The main working stages of MACAM, lines with different colors are used to represent whether the corresponding data flows occur in dense mode or sparse mode.

format into the reused delay registers. In sparse mode, after the indexes have been written, the controller will send the column indexes to arrays configured in CAM mode and perform search operations to load the vector data into the exponent registers, preparing for the subsequent computing stage.

During the computing stage in dense mode, MACAM needs to input vector data from the input buffer into the arrays and perform MAC calculations, with the intermediate computation results temporarily stored in the output buffer and the merge queue. For the computing stage in sparse mode, the controller traverses rows and determines the data range corresponding to each row based on the row counts in MCSR format. The search results are then stored in the match buffer. Subsequently, MACAM determines which rows in the MAC array are to participate in MAC operations based on the search results. Intermediate results are also temporarily stored in the output buffer and the merge queue. Since the speed of sparse mode is slower than that of dense mode, the merge stage is tightly coupled with sparse mode and occurs immediately after the sparse-mode computation. This means that once the computation for each row is completed and the intermediate computation results in sparse mode are generated, the merge stage will commence immediately. This pipelining approach masks the latency of the merge stage as much as possible.

D. Floating-point Calculation in MACAM

A delay-based approach is used to support floating-point MAC in MACAM. Fig. 7 shows the process of performing MAC calculations on matrix mantissas stored in MAC arrays. The exponents of matrices and vectors, which are stored in the exponent files, are summed using the adder in the shift-and-add unit and stored in the exponent file. The delay unit identifies the maximum value among the exponents corresponding to the data involved in the computation in the MAC array. Specifically, in dense mode, the data stored on different bit-lines is pre-aligned to the maximum exponent using a zero-padding method which is used for alignment in dense format, enabling the entire MAC array to participate in computation simultaneously. In sparse mode, only the regions corresponding to matching indexes are involved in the computation. Since the indexes are stored in a row-major order, the non-zero elements from the matched row of matrix are often stored

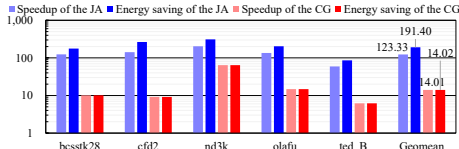


Fig. 11: Speedup and energy saving of MACAM in Different Applications.

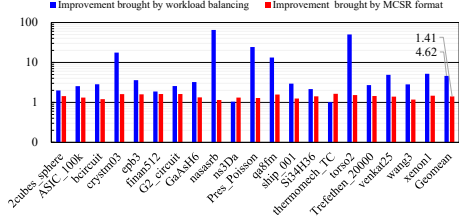


Fig. 12: Performance improvements brought by workload balancing and using MCSR format.

validating the effectiveness of combining the two modes.

For the application-level evaluation, we use MACAM to accelerate the high-precision SpMV kernels in the Jacobi iterative algorithm and the sparse conjugate gradient (CG) algorithm, while the other parts (including the preprocessing algorithm, which takes less than 10% of the overall time) are running on the GPU. As shown in Fig. 11, MACAM achieves $123.33\times$ and $14.01\times$ speedup and $191.4\times$ and $14.02\times$ energy saving over the GPU baseline in the two sparse iterative algorithms, respectively.

B. Ablation Experiment of Optimization Methods

MACAM uses MCSR format and a workload balancing approach as optimization methods. To demonstrate the effectiveness of these two optimization methods, we conducted ablation experiments on MACAM. As shown in Fig. 12, on average, MACAM with workload balancing and MACAM with the MCSR format achieve performance improvements of $4.62\times$ and $1.41\times$, respectively, compared with variants without these optimization methods. The two optimization methods are orthogonal and do not interfere with each other. The MACAM used in our aforementioned evaluations has incorporated these two optimization methods.

C. Breakdown of MACAM

Fig. 13 shows the area and power consumption breakdown of MACAM. The total area of MACAM is 23.04 mm^2 and the average running power of MACAM is 42.10 W . The ADC occupies the majority of the area and power consumption because we have provided two ADCs with different precisions for two modes, which incurs a certain area and power cost.

Fig. 14 shows the runtime breakdown of MACAM. The merging stage is tightly coupled with the sparse computing stage, and its time is also included in the sparse computing stage. The sparse computing stage takes longer than the dense computing stage, which demonstrates that the dense mode has a higher degree of computational parallelism and exhibits better performance. Additionally, the write time accounts for a relatively small proportion, proving that the slower write time of ReRAM does not have a significant impact on the overall computing process.

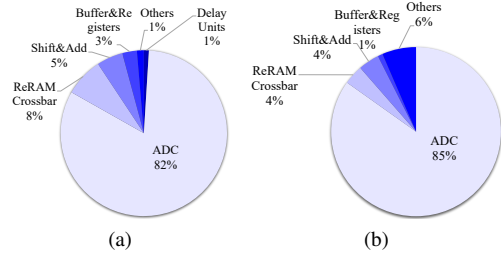


Fig. 13: Breakdown of MACAM. (a) Area. (b) Power.

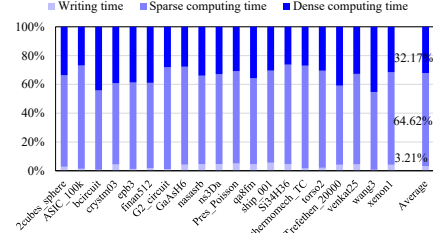


Fig. 14: Runtime breakdown of MACAM.

D. The Impact of Non-Ideal Factors of Devices

As shown in Fig. 14, the matrix programming latency has been included within MACAM’s runtime. For non-ideal programming, when operating in dense mode, the computational accuracy of the single-level cell based RRAM array is hardly affected until the programming error exceeds 5% [8]. MACAM utilizes a combination of sparse and dense modes, in which only a portion of the rows in the array are activated in sparse mode, resulting in lower computational errors compared to when only the dense mode is used. Current research indicates that the programming error of ReRAM can be controlled within 1% [25]. Under these circumstances, the results from 1000 Monte Carlo simulations indicate that the computation error of MACAM is 0.2%. In terms of lifetime evaluation, assuming the endurance of ReRAM is 10^{10} [26] cycles and the system operates continuously, MACAM can function as an SpMV accelerator for iterative solvers for approximately 3.5 years. Since MACAM integrates homogeneous multifunctional ReRAM arrays, the wear-leveling techniques [27], [28] for enhancing the lifetime of CIM accelerators can also be applied to MACAM, which will significantly extend ($100\times$ as mentioned in [28]) the lifetime of MACAM.

V. CONCLUSION

Due to the lack of flexibility, it is challenging for a fixed CIM architecture to support both high-parallel computing and irregular sparse data storage for SpMV. We propose MACAM, a flexible CIM accelerator for high-precision SpMV. Each array of MACAM can be configured to meet the storage and computing requirements of the sparse mode or dense mode. Two optimization methods are further proposed to improve the performance of MACAM. Experimental results show that MACAM achieves considerable speedup and energy saving compared with GPU baseline and two state-of-the-art CIM SpMV accelerators by exploiting the local-sparsity of sparse matrices and working in a more flexible and efficient manner.

REFERENCES

- [1] C. Aykanat, F. Ozguner, F. Ercal, and P. Sadayappan, "Iterative algorithms for solution of large sparse systems of linear equations on hypercubes," *IEEE Transactions on Computers*, vol. 37, no. 12, pp. 1554–1568, 1988.
- [2] Y. Saad and H. A. van der Vorst, "Iterative solution of linear systems in the 20th century," *Journal of Computational and Applied Mathematics*, vol. 123, no. 1, pp. 1–33, 2000, numerical Analysis 2000. Vol. III: Linear Algebra.
- [3] J. R. Gilbert, S. Reinhardt, and V. B. Shah, "High-performance graph algorithms from parallel sparse matrices," in *Applied Parallel Computing. State of the Art in Scientific Computing*, 2007, pp. 260–269.
- [4] S. Yesil, A. Heidarshenas, A. Morrison, and J. Torrellas, "Speeding up spmv for power-law graph analytics by enhancing locality & vectorization," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2020, pp. 1–15.
- [5] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Advances in Neural Information Processing Systems*, vol. 29, 2016.
- [6] M. Mahesh, S. Nalesh, and S. Kala, "Hardware acceleration of spmv multiplier for deep learning," in *2021 25th International Symposium on VLSI Design and Test (VDATE)*, 2021, pp. 1–6.
- [7] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney, "Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters," *Internet Mathematics*, vol. 6, no. 1, pp. 29–123, 2009.
- [8] B. Feinberg, U. K. R. Vengalam, N. Whitehair, S. Wang, and E. Ipek, "Enabling scientific computing on memristive accelerators," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, 2018, pp. 367–382.
- [9] L. Song, Y. Zhuo, X. Qian, H. Li, and Y. Chen, "GraphR: Accelerating Graph Processing Using ReRAM," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2018, pp. 531–543.
- [10] T. Yang, D. Li, Y. Han, Y. Zhao, F. Liu, X. Liang, Z. He, and L. Jiang, "Pimgcn: A rram-based pim design for graph convolutional network acceleration," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, 2021, pp. 583–588.
- [11] X. Zhang, Z. Li, R. Liu, X. Chen, and Y. Han, "FSPA: An FeFET-based Sparse Matrix-Dense Vector Multiplication Accelerator," in *2023 60th ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2023.
- [12] M. Fan, X. Chen, D. Yang, Z. Jin, and W. Liu, "Recg: Rram-accelerated sparse conjugate gradient," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, 2024.
- [13] X. Zhang, Z. Li, R. Liu, X. Chen, and Y. Han, "Gas: General-purpose in-memory-computing accelerator for sparse matrix multiplication," *IEEE Transactions on Computers*, vol. 73, no. 6, pp. 1427–1441, 2024.
- [14] G. Gerogiannis, S. Ananthakrishnan, J. Torrellas, and I. Hur, "Hottiles: Accelerating spmm with heterogeneous accelerator architectures," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2024, pp. 1012–1028.
- [15] D. Reis, A. F. Laguna, M. Niemier, and X. S. Hu, "Attention-in-Memory for Few-Shot Learning with Configurable Ferroelectric FET Arrays," in *2021 26th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2021, pp. 49–54.
- [16] X. Zhang, R. Liu, T. Song, Y. Yang, Y. Han, and X. Chen, "Re-FeMAT: A Reconfigurable Multifunctional FeFET-based Memory Architecture," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2022.
- [17] N. Challapalle, S. Rampalli, L. Song, N. Chandramoorthy, K. Swaminathan, J. Sampson, Y. Chen, and V. Narayanan, "GaaS-X: Graph Analytics Accelerator Supporting Sparse Data Representation using Crossbar Architectures," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020, pp. 433–445.
- [18] S. Feng, J. Sun, S. Pal, X. He, K. Kaszyk, D.-h. Park, M. Morton, T. Mudge, M. Cole, M. O'Boyle, C. Chakrabarti, and R. Dreslinski, "Cospars: A software and hardware reconfigurable spmv framework for graph analytics," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, 2021, pp. 949–954.
- [19] W. Kahan, "IEEE standard 754 for binary floating-point arithmetic," *Lecture Notes on the Status of IEEE*, vol. 754, no. 94720-1776, p. 11, 1996.
- [20] Y. S. Chen, H. Y. Lee, P. S. Chen, P. Y. Gu, C. W. Chen, W. P. Lin, W. H. Liu, Y. Y. Hsu, S. S. Sheu, P. C. Chiang, W. S. Chen, F. T. Chen, C. H. Lien, and M.-J. Tsai, "Highly scalable hafnium oxide memory with improvements of resistive distribution and read disturb immunity," in *2009 IEEE International Electron Devices Meeting (IEDM)*, 2009, pp. 1–4.
- [21] ASU, "Predictive Technology Model," 2011.
- [22] K. D. Choo, J. Bell, and M. P. Flynn, "27.3 Area-efficient 1GS/s 6b SAR ADC with charge-injection-cell-based DAC," in *2016 IEEE International Solid-State Circuits Conference (ISSCC)*, 2016, pp. 460–461.
- [23] L. Kull, T. Toifl, M. Schmatz, P. A. Francese, C. Menolfi, M. Braendli, M. Kossel, T. Morf, T. M. Andersen, and Y. Leblebici, "A 3.1mw 8b 1.2gs/s single-channel asynchronous sar adc with alternate comparators for enhanced speed in 32nm digital soi cmos," in *2013 IEEE International Solid-State Circuits Conference Digest of Technical Papers*, 2013, pp. 468–469.
- [24] T. A. Davis and Y. Hu, "The University of Florida Sparse Matrix Collection," *ACM Trans. Math. Softw.*, vol. 38, no. 1, dec 2011.
- [25] B. Q. Le, A. Levy, T. F. Wu, R. M. Radway, E. R. Hsieh, X. Zheng, M. Nelson, P. Raina, H.-S. P. Wong, S. Wong, and S. Mitra, "Radar: A fast and energy-efficient programming technique for multiple bits-per-cell rram arrays," *IEEE Transactions on Electron Devices*, vol. 68, no. 9, pp. 4397–4403, 2021.
- [26] Y. Y. Chen, B. Govoreanu, L. Goux, R. Degraeve, A. Fantini, G. S. Kar, D. J. Wouters, G. Groeseneken, J. A. Kittl, M. Jurczak, and L. Altissime, "Balancing set/reset pulse for $> 10^{10}$ endurance in HfO₂/Hf 1t1r bipolar rram," *IEEE Transactions on Electron Devices*, vol. 59, no. 12, pp. 3243–3249, 2012.
- [27] H. Zhou, B. Wu, H. Cheng, W. Zhao, X. Wei, J. Liu, D. Feng, and W. Tong, "Odpim: A write-optimized and long-lifetime rram-based accelerator for online deep learning," in *2023 Design, Automation & Test in Europe Conference Exhibition (DATE)*, 2023, pp. 1–6.
- [28] H. Zhou, B. Wu, H. Cheng, J. Liu, T. Lei, D. Feng, and W. Tong, "Drctl: A disorder-resistant computation translation layer enhancing the lifetime and performance of memristive cim architecture," in *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2024, pp. 263–277.