HPN-SpGEMM: Hybrid PIM-NMP for SpGEMM

Kwangrae Kim and Ki-Seok Chung

***Abstract*—Sparse matrix-matrix multiplication (SpGEMM) is widely used in various scientific computing applications. However, the performance of SpGEMM is typically bound by memory performance due to irregular access patterns. Prior accelerators leveraging high-bandwidth memory (HBM) with optimized data flows still face limitations in handling sparse matrices with varying sizes and sparsity levels. We propose HPN-SpGEMM, a hybrid architecture that employs both processing-in-memory (PIM) cores inside bank groups and near-memory-processing (NMP) cores in the logic die of an HBM memory. To the best of our knowledge, this is the first hybrid architecture for SpGEMM that leverages both PIM cores and NMP cores. Evaluation results demonstrate significant performance gains, effectively overcoming memory-bound constraints.**

***Index Terms*—SpGEMM, Processing-in-Memory, Near-Memory-Processing, HBM.**

I. Introduction

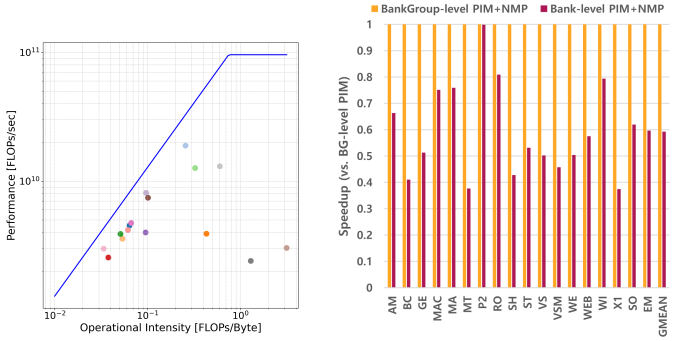
Sparse matrix-matrix multiplication (SpGEMM) is a key kernel widely used in various scientific domains [9]. However, the performance of SpGEMM is typically limited by memory performance due to its irregular access patterns. Previous accelerators have attempted to address these limitations through various hardware optimizations leveraging high-bandwidth memory (HBM). They have adopted different SpGEMM data flows, such as inner-product, outer-product, or row-by-row approaches, and accelerators were designed in a way to optimize for those specific data flows to enhance performance [1]. However, they often failed to accelerate SpGEMM with various dimensions and sparsity levels. As observed in the roofline analysis in Fig. 1a, GAMMA [1], a widely known accelerator employing a row-by-row approach, shows memory-bound characteristics. This is primarily due to the many memory commands required to carry out irregular accesses for input matrices and intermediate partial sums. These observations strongly motivate the need to develop a novel HBM-based processing-in-memory (HBM-PIM) architecture to accelerate SpGEMM.

In this letter, we propose a novel hybrid HBM-based architecture called *HPN-SpGEMM*. The architecture integrates bank group (BG)-level PIM cores on the DRAM dies and pseudo-channel (pCH)-level near-memory processing (NMP) cores on the logic die. It accelerates SpGEMM by parallelizing the partial sum generation in the BG-PIM cores and the result aggregation in the pCH-NMP cores. To further enhance this parallel execution, the data flow is designed so that the memory command data paths for BG-PIM do not conflict with those for the pCH-NMP cores. Additionally, we propose an optimized input matrix mapping strategy for effective load balancing.

This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (RS-2024-00409492)

Kwangrae Kim and Kiseok Chung are with the Department of Electronic Engineering, Hanyang University, Seoul, 04763, South Korea (e-mail: kksilver91@hanyang.ac.kr; kchung@hanyang.ac.kr)

0000-0000 © 2024 IEEE



(a) (b)

Fig. 1: a) Roofline analysis of GAMMA [1], b) Performance comparison between Bank-level PIM+NMP and BG-level PIM+NMP. All evaluations are conducted on matrices listed in Table I.

II. Design Motivation

1. Hybrid PIM-NMP Architecture for SpGEMM: While conventional PIM architectures often adopt bank-level PIM to exploit high parallelism [10], [13], this approach is not suitable for SpGEMM because aggregating partial sums must be frequently conducted, which incurs significant communication complexity and control overhead [11]. To overcome this limitation, we propose a hybrid PIM-NMP architecture that distributes the processing loads across two hierarchy levels: 1) On the logic die, pCH-level NMP cores aggregate partial sums across multiple pseudo-channels (pCHs) via TSVs, and 2) On the DRAM die, PIM cores generate partial sums utilizing high internal bandwidth. However, assigning PIM units to lower hierarchy levels (e.g., bank-level) introduces a trade-off. As the granularity becomes finer, the size of submatrices allocated to each unit decreases, making load balancing more difficult. This issue becomes even more pronounced in SpGEMM, which exhibits irregular memory access patterns. As shown in Fig. 2b, bank-level PIM suffers from performance degradation due to load imbalance, achieving only 59% of the performance of BG-level PIM + NMP on average. Based on this result, we adopt a hybrid structure that combines pCH-level NMP with BG-level PIM, which is a higher hierarchy than bank-level, to process SpGEMM workloads efficiently.

2. Row-by-Row Method for SpGEMM: The row-by-row method of SpGEMM computes the output row $C[i,*]$ by iterating over the nonzero elements a_{ik} in the input row A_i, and multiplying each a_{ik} with the corresponding row $B[k,*]$,

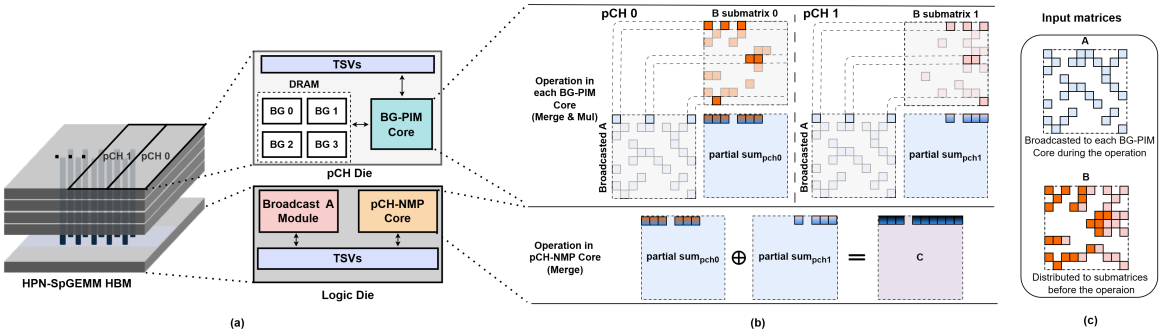


Fig. 2: Overview of HPN-SpGEMM where the number of pseudo channels (pCHs) is two and the number of bank groups (BG) is four: a) Overall architecture with key modules, b) Operations in each hardware unit, illustrating the process of generating partial sums and merging them for each row of A, c) Description of input matrices.

and summing the results (i.e., $C[i,*] = \sum_{k=0}^{N} A[i,k] \times B[k,*]$). Compared to the other methods, the row-by-row method achieves a better trade-off between reuse efficiency and computational complexity [1]. Therefore, we adopt this row-by-row method, modifying the method to fit our proposed architecture.

III. HPN-SpGEMM

A. Overview of HPN-SpGEMM

Fig. 2 illustrates the overall architecture with the key modules and their operations of the HPN-SpGEMM for SpGEMM ($A \times B = C$). Both input matrices, A and B, are stored in the compressed sparse row (CSR) format [8], and the result matrix C is also generated in the same format. Leveraging the hierarchical structure of HBM (pCHs and BGs), HPN-SpGEMM implements BG-level PIM cores inside each pCH die and pCH-level NMP cores on the logic die. Since elements of A are accessed sequentially and needed for generating all partial sums on a row-by-row basis (Fig. 2b), it must be simultaneously provided to all BG-PIM cores. Thus, A is dynamically broadcast during the computation by the *Broadcast A* module in the logic die. In contrast, B is pre-distributed across pCHs with a specific mapping method so that BG-PIM cores can access B submatrices locally. The BG-PIM cores generate partial sums by merging and multiplying broadcasted A with the distributed B submatrices. The pCH-NMP cores then aggregate the partial sums from each BG-PIM core to generate the final C.

B. Hardware architecture

Hardware configuration on the logic die (Fig. 3a, Fig. 3b): The pCH-NMP core and the *Broadcast A* module exploit pCH-level parallelism (with #pCH data paths) to issue memory commands for generating partial sums and broadcasting A, respectively (Fig. 3b). These commands are then scheduled by the *Central memory request arbiter* module before being sent to DRAM. The *addition processing element* (APE) scheduler in the pCH-NMP core continuously polls the *multiplication processing element* (MPE) scheduler on the BG-PIM core to track which partial sums have been generated from the BG-PIM cores. The APE scheduler assigns accumulation tasks to

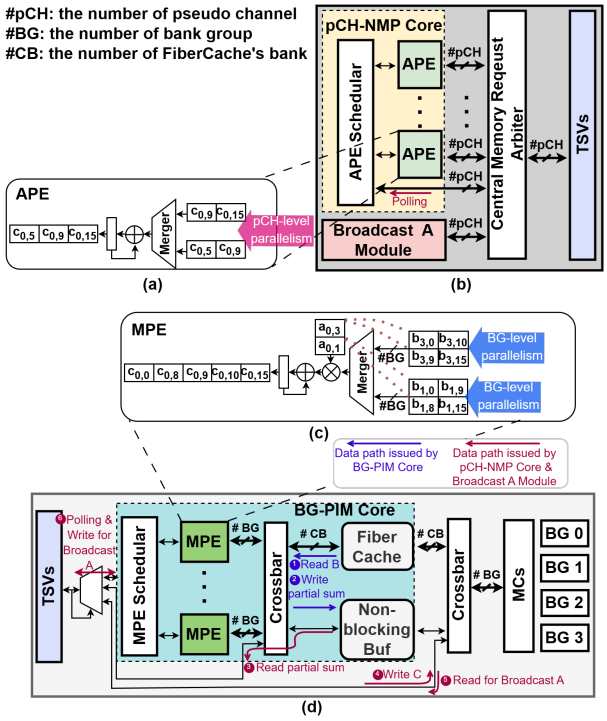


Fig. 3: Description of HPN-SpGEMM's detailed internal architecture: a) APE (example with #pCH = 2), b) Logic die of HPN-SpGEMM, c) MPE (example with #BG = 2, capable of merging two A rows), d) pCH die of HPN-SpGEMM.

APEs based on the tracked data. APEs accumulate the received partial sums in parallel to generate the final C matrix and store the final result in DRAM. Fig. 3a depicts the process in which the APE receives partial sums from each pCH, utilizing pCH-level parallelism, and generates the final product C ($\{c_{0,9}, c_{0,15}\}+\{c_{0,5}, c_{0,9}\} \Rightarrow \{c_{0,5}, c_{0,9}, c_{0,15}\}$). The radix of the APE's merger is equal to the number of pCHs (#pCH). The memory commands issued within the NMP core (broadcast, polling, partial sum read/write) are scheduled sequentially by a finite state machine (FSM) we design to avoid internal data traffic contention.

Hardware configuration on the pCH die (Fig. 3c, Fig. 3d): The MPE scheduler assigns tasks to each MPE based on the

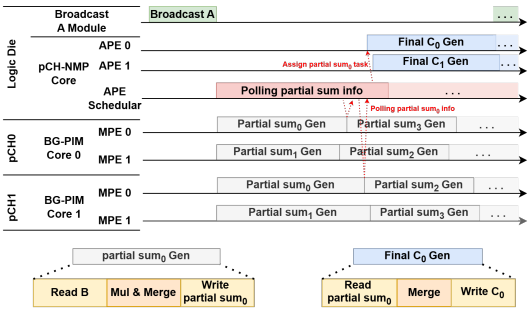


Fig. 4: Execution flow of HPN-SpGEMM (example with two APEs and two MPEs per module, #pCH = 2)

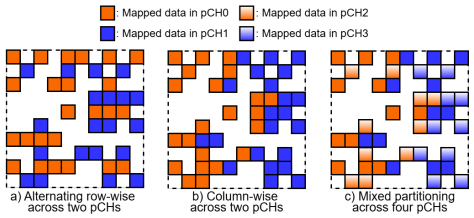


Fig. 5: Description of mapping method

broadcasted A. Each MPE executes partial sum generation (Partial Sum Gen) through the following process: 1) fetching the corresponding rows of the B's submatrix in BG-level parallel; 2) performing multiplication & merge operations with broadcasted A; and 3) writing the computed partial sums to the non-blocking buffer. Fig. 3c illustrates the process in which an MPE fetches the corresponding rows of B and generates a partial sum ($a_{0,3} \cdot \{b_{3,0}, b_{3,9}, b_{3,10}, b_{3,15}\} + a_{0,1} \cdot \{b_{1,0}, b_{1,8}, b_{1,9}, b_{1,15}\} \Rightarrow \{c_{0,0}, c_{0,8}, c_{0,9}, c_{0,10}, c_{0,15}\}$). In case the number of rows in A exceeds the radix of the MPE merger, the MPE writes the intermediate partial sum to a cache called FiberCache [1] and later reads it back to perform the remaining merge operation, completing the partial sum generation.

A key design challenge in the BG-PIM core is ensuring that the memory command paths of the logic die and the BG-PIM core do not overlap. Overlapping paths may cause frequent memory access conflicts, hindering the parallel execution of the partial sum generation and the final C generation. To avoid this issue, the proposed architecture partitions on-chip storage into FiberCache and a non-blocking buffer. FiberCache —adopted from GAMMA [1]— efficiently manages the intermediate partial sums and non-zero elements and provides quick access for multiple processing elements (PEs). The non-blocking buffer stores partial sums that exhibit no temporal locality, so memory access can continue without stalls even when a cache miss occurs. As shown in Fig. 3d, both storage components connect through crossbars to effectively isolate the memory commands issued from the BG-PIM core (❶, and ❷) from those issued by the Broadcast A module and the pCH-NMP core (❸, ❹, ❺, and ❻). Due to the scheduling of the pCH-NMP core, command conflicts ❷ and ❸ do not occur.

Fig. 4 illustrates the execution flow of the units on the logic die and the pCH die, demonstrating how the broadcast of A, BG-level partial sum generation on the pCH die, and final computation of C on the logic die are performed concurrently.

C. Mapping Strategy for SpGEMM

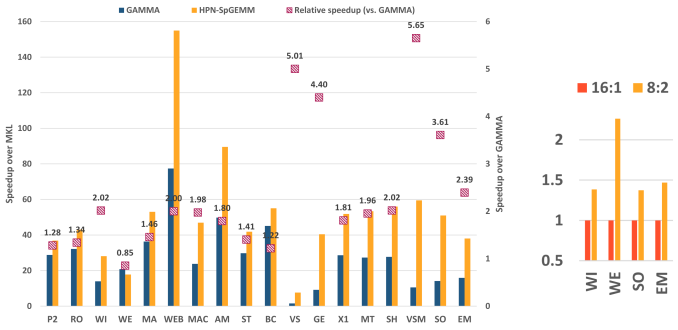
In the row-by-row SpGEMM method, non-zero elements in matrix A are accessed sequentially, so matrix A is mapped across multiple pCHs following the conventional HBM physical mapping scheme. In contrast, matrix B exhibits irregular access patterns, and thus, the strategy for distributing B across multiple pCHs significantly affects load balance and the overall performance. To address this issue, an alternating row-wise mapping scheme may be considered. This approach distributes rows evenly across pCHs in a round-robin manner and simplifies row index reconstruction (Fig. 5a). However, if non-zero elements are unevenly distributed across rows, load imbalance may still arise. This issue is particularly exacerbated when the coefficient of variation (CV, standard deviation divided by mean) of non-zero counts per row is high (e.g., CV $\geq$ 2), which indicates significant row-wise load imbalance. To mitigate this, we propose a *mixed partitioning* approach (Fig. 5c) that partially incorporates column-wise partitioning into the row-wise scheme, aiming to further distribute non-zero elements more evenly at the row level. While this strategy introduces slight overhead in row pointer management, the impact is negligible relative to the overall matrix size. In our implementation, we apply the mixed strategy with an 8:2 ratio — performing eight alternating row-wise partitions followed by two column-wise partitions - to match the 16 pCH architecture.

IV. Evaluation

Experimental Setup: We evaluate the performance of HPN-SpGEMM and GAMMA [1], a representative SpGEMM accelerator, using Ramulator [5] with a common memory configuration of 128 GB/s HBM [6] featuring 16 pCHs per DRAM die (each with 16 banks and 4 BGs). GAMMA is configured with 48 32-radix PEs and a 1.5 MB FiberCache with 48 cache banks. For HPN-SpGEMM, a total of 32 32-radix MPEs are deployed across all 16 pCH dies (with 2 MPEs per pCH die), where each 32-radix MPE receives input through a 4-radix merger leveraging BG-level parallelism. The logic die is equipped with 16 16-radix APEs. A total of 0.75 MB of FiberCache and 0.75 MB of non-blocking buffer are allocated evenly across all 16 pCH dies, so each pCH die has a 46.875 KB FiberCache and a 46.875 KB non-blocking buffer. The FiberCache in each pCH die is structured with 8 cache banks to support BG-level access for the 2 MPEs. All designs operate with a system clock of 1 GHz. We also compare SpGEMM performance with Intel MKL's mkl_sparse_spmm [7] on a

TABLE I: Sparse matrices from SuiteSparse [2] for evaluation

Benchmark	Dimensions	Sparsity	Benchmark	Dimensions	Sparsity
p2p-Gnutella31 (P2)	62,586	3.8E-05	amazon-2008 (AM)	735,323	9.5E-06
roadNet-TX (RO)	1,393,383	2.0E-06	stomach (ST)	213,360	6.6E-05
wiki-talk-temporal (WI)	1,140,149	2.5E-06	bcsstk32 (BC)	44,610	1.0E-03
webbase-1M (WE)	1,000,005	3.1E-06	vsp_bcsstk30_500sep_10in_1Kout (VS)	58,348	1.2E-03
m_t1 (MT)	97,578	1.0E-03	Ge99H100 (GE)	112,985	6.6E-04
web-Google (WEB)	916,428	6.1E-06	x104 (X1)	108,384	7.4E-04
mario002 (MA)	389,874	1.4E-05	vsp_msc10848_300sep_100in_1K (VSM)	21,996	5.1E-03
mac_econ_fwd500 (MAC)	206,500	3.0E-05	ship_001 (SH)	34,920	3.2E-03
soc-Slashdot0811 (SO)	77,360	1.5E-04	email-Enron (EM)	36,692	2.7E-04



(a) (b)

Fig. 6: Performance evaluation results: a) Speedup comparison of GAMMA and HPN-SpGEMM over MKL (left Y-axis) and the relative speedup of HPN-SpGEMM over GAMMA (right Y-axis), b) Performance comparison based on the mapping method for HPN-SpGEMM.

server with an Intel Core i9-10900X and 8 DDR4-2666. GPU results are not included as the SpGEMM execution on the GPU showed similar performance to MKL [1]. We use sparse matrices from SuiteSparse [2] to evaluate the performance (Table I). The matrices are stored in the CSR format [8], with values represented as double-precision floating points (8B) and row pointers as 4B integers.

Performance Analysis: Fig. 6a presents the performance comparison of GAMMA and HPN-SpGEMM relative to MKL. HPN-SpGEMM achieves an average speedup of 48.73× (up to 154.99×) over MKL. Compared to GAMMA, HPN-SpGEMM demonstrates an average speedup of 2.22× (up to 5.65×). For the WE matrix, GAMMA slightly outperforms HPN-SpGEMM due to high cache locality (hit rate: 0.86). However, for matrices such as VS, GE, and VSM, GAMMA exhibits low cache utilization (0.05, 0.12, and 0.04 hit rates), resulting in considerably lower performance (achieving speedups of only 1.53×, 9.18×, and 10.52× over MKL). In contrast, HPN-SpGEMM demonstrates significantly higher performance, achieving speedups of 7.64×, 40.41×, and 59.47× over MKL for VS, GE, and VSM. This performance gain is attributed to the proposed architecture and operation flow of HPN-SpGEMM, which effectively mitigates memory-bound issues by efficiently utilizing internal memory bandwidth.

Mapping Method Analysis: Fig. 6b compares the performance of the 8:2 mixed partitioning scheme and the 16:1 row-wise partitioning scheme, with a focus on matrices whose coefficient of variation (CV) of non-zero elements per row exceeds 2. For matrices with low CV (ranging from 0.03 to 1.26) that are not shown in Fig. 6b, both partitioning schemes yield comparable performance, with differences ranging from 0.9× to 1.07×. In the case of the ST matrix, the 8:2 scheme shows a 27% performance drop compared to 16:1; however, it still achieves a 1.41× improvement over the GAMMA baseline, demonstrating the robustness of the proposed method. On the other hand, matrices with high CVs—WI (CV: 58.10), WE (CV: 8.16), SO (CV: 3.15), and EM (CV: 3.60)—exhibit highly uneven distributions of non-zero elements across rows. In such cases, the proposed 8:2 mixed mapping scheme outperforms the 16:1 scheme by 1.41×, 2.43×, 1.37×, and 1.47×, respectively. Although the 8:2 scheme introduces a slight storage overhead due to additional row pointers (ranging from 1.002× to 1.096×), this overhead is negligible.

Design Overhead: APEs and MPEs were implemented in Verilog HDL and synthesized using the 28 nm Nangate Open-Cell library with Synopsys Design Compiler. Both designs operate at 1GHz. To reflect realistic fabrication, the units in the logic die were scaled to a 20 nm CMOS process, where 16 APEs occupy 1.22 mm², corresponding to 1.43% of the total logic die area (85.4 mm²). For the units in the pCH die, we applied 20 nm scaling followed by a 10× area increase to account for DRAM-specific constraints such as lower logic density [12]. The FiberCache and the non-blocking buffer were modeled using CACTI [3], and the crossbar was modeled based on [4]. The total area of the pCH components —FiberCache, non-blocking buffer, crossbar, and MPE— occupies 5.10 mm², which corresponds to 12.1% of the pCH die area (42.15 mm²).

V. Conclusion

HPN-SpGEMM efficiently accelerates SpGEMM by integrating BG-level PIM and pCH-level NMP. Its architecture mitigates memory-bound constraints through efficient data mapping and execution flow. Evaluation results show significant speedups over a widely known accelerator GAMMA, demonstrating its effectiveness in handling large, sparse matrices.

References

[1] G. Zhang *et al.*, "Gamma: Leveraging Gustavson's algorithm to accelerate sparse matrix multiplication," in *Proc. ACM Int. Conf. Arch. Support Program. Lang. Oper. Syst. (ASPLOS)*, 2021, pp. 687–701.
[2] T. A. Davis *et al.*, "The University of Florida sparse matrix collection," *ACM Trans. Math. Softw.*, vol. 38, no. 1, pp. 1–25, 2011.
[3] R. Balasubramonian *et al.*, "CACTI 7: New tools for interconnect exploration in innovative off-chip memories," *ACM Trans. Arch. Code Optim.*, vol. 14, no. 2, pp. 1–25, 2017.
[4] K. Sewell *et al.*, "Swizzle-switch networks for many-core systems," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 2, no. 2, pp. 278–294, 2012.
[5] Y. Kim *et al.*, "Ramulator: A fast and extensible DRAM simulator," *IEEE Comput. Arch. Lett.*, vol. 15, no. 1, pp. 45–49, 2015.
[6] "JEDEC Publishes HBM2 Specification," 2016. [Online]. Available: http://www.anandtech.com/show/9969/jedec-publisheshbm2-specification
[7] Intel math kernel library, *High-Performance Computing on the Intel® Xeon Phi™*, Springer, Cham, 2014.
[8] "Compressed Sparse Row matrix (CSR)," [Online]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.csr_matrix.html. Accessed: Mar. 12, 2025.
[9] S. Han *et al.*, "EIE: Efficient inference engine on compressed deep neural network," *ACM SIGARCH Comput. Arch. News*, vol. 44, no. 3, pp. 243–254, 2016.
[10] S. Lee *et al.*, "Hardware architecture and software stack for PIM based on commercial DRAM technology: Industrial product," *Proc. ACM/IEEE 48th Annu. Int. Symp. Comput. Archit. (ISCA)*, pp. 43–56, 2021
[11] G. Jonatan *et al.*, "Scalability limitations of processing-in-memory using real system evaluations," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 8, no. 1, Art. 5, pp. 1–28, Mar. 2024.
[12] F. Devaux, "The true processing in memory accelerator," in *Proc. IEEE Hot Chips 31 Symp. (HCS)*, pp. 1–24, 2019.
[13] X. Xie *et al.*, "SpaceA: Sparse matrix vector multiplication on processing-in-memory accelerator," in *Proc. IEEE Int. Symp. High-Performance Computer Architecture (HPCA)*, 2021, pp. 570–583.