Trends and Opportunities for SRAM Based In-Memory and Near-Memory Computation

Srivatsa Srinivasa[1], Akshay Krishna Ramanathan[2], Jainaveen Sundaram[1], Dileep Kurian[1], Srinivasan Gopal[1], Nilesh Jain[1], Anuradha Srinivasan[1], Ravi Iyer[1], Vijaykrishnan Narayanan[2] and Tanay Karnik[1]

[1] Intel Corporation [2] The Pennsylvania State University.

Abstract

Changes in application trends along with increasing number of connected devices have led to explosion in the amount of data being generated every single day. Computing systems need to efficiently process these huge amounts of data and generate results, classify objects, stream high quality videos and so on. In-Memory Computing and Near-Memory Computing have been emerged as the popular design choices to address the challenges in executing the above-mentioned tasks. Through In-Memory Computing, SRAM Banks can be repurposed as compute engines while performing Bulk Boolean operations. Near-Memory techniques have shown promise in improving the performance of Machine learning tasks. By carefully understanding the design we describe the opportunities towards amalgamating both these design techniques for obtaining further performance enhancement and achieving lower power budget while executing fundamental Machine Learning primitives. In this work, we take the example of Sparse Matrix Multiplication, and design an I-NMC accelerator which speeds up the index handling by 10x-60x and 10x-70x energy efficiency based on the workload dimensions as compared with non I-NMC solution occupying just 1% of the overall hardware area.

Keywords

In-Memory Computation, Near-memory Computation, SRAM

1. Introduction

With the increasing data centric nature of present-day application domains like AI, graph analytics etc., speed of execution and energy efficiency challenges have become utmost importance metrics. Current computing techniques limits performance and spend huge amount of energy in executing these data centric tasks. Traditional systems require huge amount data to be transferred from a memory system to the processing elements (PEs) through a very narrow and parasitic channel resulting in a performance bottleneck. Increase in overall compute capacity with additional hardware and PEs will become redundant if the channel BW is not able to provide the required data at the right time. While memories rely on capacity aware design (packing more bits on chip), PEs favor performance aware design. Capacity centric design makes the memories slower, bulkier and performance centric design makes the PEs more complex, area inefficient and power hungry.

Blurring the ever-growing logic-memory performance gap (due to contrasting design requirements) has led to revisiting the already existing design strategies and look for alternate and yet promising solutions. Multiple published works from the literature emphasize on various caching and tagging strategies for addressing the performance gap in processors [1,2]. GPUs have been shown to accelerate the performance of many data centric workloads, at the cost of high-power consumption. FPGAs are capable of providing a high degree of parallelism but running at reduced clock frequencies and thereby compromising on the speed of AI/ML execution.

Another class of design solution gaining wide popularity is to compute wherever the data resides or by designing PEs near the memory system. Subcategories in this class are In-Memory Computing (IMC) and Near Memory Computing (NMC). In this paper, we term IMC for those techniques which manipulate data as part of memory access without needing a conventional hardware. We term those techniques as NMC, which require a standard hardware integrated at the periphery of memory system with wide direct access to the memory. Advantages of IMC and NMC are attributed to two main design principles of the memory system, a) hierarchical organization into Banks and sub-arrays and b) Huge Bandwidth (BW) internal to the system. Hierarchical organization provides massive computation parallelism while high internal BW minimizes the data movement. By taking advantage of these two properties effectively, IMC and NMC techniques provide higher performance at a fraction of power budget.

The compute centric to memory centric design evolution through IMC and NMC are further fueled by unique properties (non-volatility, crossbar architecture, high density etc..) of emerging memory technologies [3-5] (FeFET, STT-MRAM, PCM etc..) and developments in 3D integration process (HBM, HMC, sequential 3D integration etc..). However, high write latency, low endurance and operations susceptible to variations at low voltage make these emerging memories not suitable candidates to replace on chip SRAM memories in the near future.

Hence numerous solutions have been proposed to enable IMC and NMC with SRAM as the target memory system. Most of the IMC solutions propose to minimally alter the SRAM cell and read technique to perform computation without drastically affecting the memory density. Some solutions make use of monolithic 3D integration [6] to enhance the computation capability of SRAM. However, IMC cannot perform all the fundamental and performance critical tasks as part of data readout. Hence NMC technique or combination of IMC and NMC will bring accelerated performance to many application categories. This paper describes the trends and opportunities for SRAM based IMC and NMC techniques. As an example, we will also describe effectiveness of combining IMC and NMC while designing hardware for Sparse Matrix Multiplication (SpGeMM).

978-1-7281-7641-3/21/$31.00 ©2021 IEEE

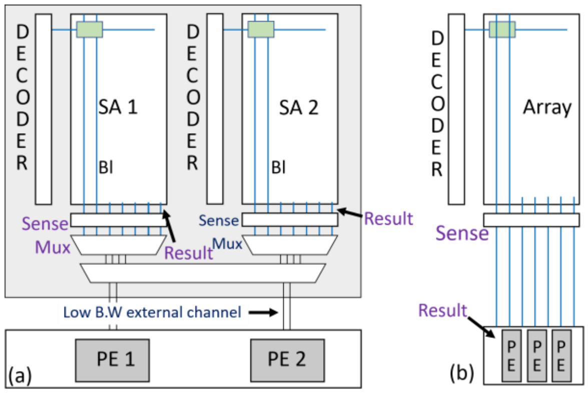

Figure 1: Illustration of the memory organization in (a) In-Memory computing design. (b) Near-Memory computing design

2. Background

2.1 IMC and NMC: Classification

Figure 1 illustrates memory organization and the channel connectivity to PEs with IMC and NMC designs. Primary characteristic of IMC designs (Figure 1(a)), is to avoid data traversal to low BW channel. Minimal modifications to the bit-cells, Peripherals like Decoder and Sense circuits are carried out in order to perform computation as part of data read out. Therefore, only a handful of computations are possible with this technique without compromising the memory properties like access speed and density.

On the other hand, PEs can directly access the High BW memory channel in case of NMC (fig. 1(b)). Even though PEs are responsible for overall computation, data mapping within the memory arrays is crucial for performance. Several studies show promise with NMC designs [7] making use of IMC to prepare the initial data for computations in PEs. Subsequent sections describe more about this strategy.

2.2 Performance metrics

Parallelism, computation cycles and area of additional logic (apart from memory) are the three basic metrics which can determine whether IMC or NMC is suitable for a particular task. For example, IMC solutions are best if implementing only Boolean logic, because of minimal changes to memory design and single cycle execution. However, for complex operation (like Multiplication), a formal definition of a metric is necessary to develop a best design strategy. Processing-in-Memory Operations per cycle (PIM_OPC) was introduced [7] for this purpose. PIM_OPC is an indicator of the degree of compute parallelism supported by the memory. For example, considering column multiplexing of 4:1, for the memory in Figure 2, 8 Boolean operations (8-bit operands and bit-parallel computing) per cycle are possible in one PIM cycle without transferring the data to PEs outside the memory. Hence PIM_OPC in this case is 8 considering all 64 bitlines computing in parallel. Higher PIM_OPC suggests higher parallelism and faster computation. Two ways of improving the PIM_OPC are a) increasing the computation parallelism and b) reducing compute cycles. We discuss this in detail for different IMC designs in the subsequent section to describe the IMC compatibility and/or need for NMC based solution.

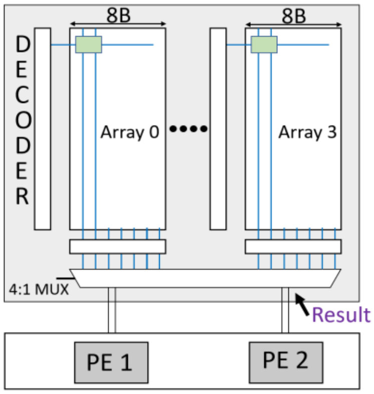

Figure 2: Illustration of PIM_OPC for IMC Boolean operation

3. Trends and opportunities: IMC

Since caches occupy significant real estate of the processor and IMC transforms the memory into thousands of in-memory accelerators, IMC techniques achieve performance much higher than CPU and GPU running bulk Boolean tasks. IMC in SRAM based processors are categorized into bitline computing which perform computation as part of data discharge pattern. Fu-Kuo Hsueh et al. [8] show Boolean operations as part of data readout with a specialized 9T SRAM. S.Aga et al.[9] proposed compute caches which repurpose the cache for computational purposes. Boolean operations are performed using multi row activation technique. Slow compute time as compared to data access as a measure to retain the memory robustness. Therefore PIM_OPC is slightly lower. Configurable SRAM cells using Monolithic 3D (M3D) integration process help shorten the cycle time and thereby reducing the denominator if PIM_OPC calculation. Work from Srinivasa et al. [10] show that the PIM_OPC can be increased up to 44% with specialized M3D SRAM cell.

In-memory addition can be performed through bitline computing by repeatedly computing the fundamental Boolean operations which are part of addition. This leads to activating the bitlines multiple times and hence increases the compute cycle (Proportional to bitline charging-discharging count). In-Memory addition operation can be broken down into three fundamental operations: a) Bit level SUM calculation b) Bit level CARRY computation and c) CARRY propagation through memory columns. While direct IMC technique can be applied for first two operations, serial CARRY propagation requires multiple cycles. This suggests that wider data requires more compute cycles.

For example,

PIM_OPC (32-bit,64 additions in parallel) =64/32 → 2

PIM_OPC (64-bit,64 additions in parallel) =64/64 → 1

In-memory addition with data wider than 64 bit brings the PIM_OPC to less than 1. However, only benefit through IMC is by offloading numerous such operations to every array of the SRAM memory and increasing the overall throughput. Moreover, having dedicated adder hardware near memory enables fast NMC at the cost of additional area. This discussion can be further extended to multiplication operations (fundamental ML computation).

Neural cache [11] and Duality cache [12] run neural networks on repurposed caches using multi row activation and bit serial computing. Both the designs achieve performance much higher than CPU and GPU running the same application. NMC techniques discussed in the next section overcome poor PIM_OPC for multiplication operations thorough NMC.

Jintao Zhang et al. [13] proposed multi row activation-based IMC for linear classification by multiplications using bitline current summation technique. This technique requires building Digital to Analog converters (DAC) and current sense amplifiers within the memory and thereby making the resultant system less dense and less robust.

While IMC shows clear benefit over CPU and GPUs while computing Boolean, arithmetic, Multiplication and classification tasks, we can clearly observe that PIM_OPC can be further improved either by NMC and/or combination of IMC and NMC techniques. Next section discusses the opportunities for improved performance while computing these tasks through NMC techniques.

4. Trends and opportunities: NMC

NMC systems perform better compared to IMC involving serial propagation and multiple cycle requirement (addition and multiplication operations which need carry propagation) mainly due to design flexibility. For example, the multiplication needs to be broken down into multiple additions and in turn the addition operation need to be broken down into multiple Boolean operations.

Many NMC technique-based accelerators have been proposed to run neural network applications efficiently. Eyeriss [14] maximizes re-use of the inputs and minimizes the partial sum reduction costs with systolic dataflow. Y. Chen et.al., [15] proposed a neural network supercomputer which maps specialized logic of the DNNs to multiple chips/nodes which are tightly intercoupled for optimizing data movement. Similarly, Simba [16] maps these operations onto multiple smaller chiplets in distributed fashion. These accelerators incur high H-tree interconnect bus penalties while fetching the data from the cache.

To avoid interconnect bus penalties, many works [7,17] have focused onto building systems which amalgamates IMC and NMC techniques. This kind of system avoids the interconnect penalties by placing the compute units close to each smallest memory unit in a SRAM memory. We categorize this type of technique as In-near memory compute (I-NMC). A.K. Ramanathan et.al.,[7] have proposed systems incorporating I-NMC technique by placing specialized compute units near SRAM arrays.

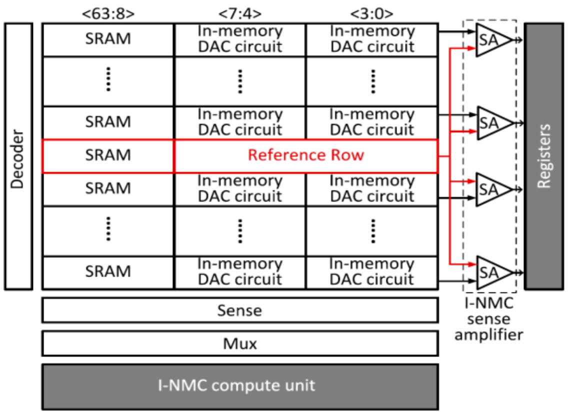

Figure 3: I-NMC technique based sparse matrix index checking engine placed in a sub-array.

Also, the I-NMC technique greatly benefits in terms of energy since the data is once read-out of the SRAM array and then computed with minimal logic energy consumption. Whereas using IMC technique, the bitlines need to be charged and discharged repetitively when computing. The bitlines are high parasitic lines connecting to the SRAM cells in an array, therefore it needs to be accessed minimally. Therefore, the energy consumption of the IMC technique is comparatively higher than the I-NMC technique when computing operations with serial propagation. For example, to compute an 8-bit multiplication, IMC requires 102 bitline (dis)charging, and I-NMC requires 8 bitline (dis)charging and the compute logic energy which is much lesser than the bitline (dis)charging energy.

Now, let us look at the sparse matrix index checking system shown in Figure 3 that uses a more enhanced version of I-NMC technique, incorporating more in-memory based compute support. Figure 3 shows a sub-array with two partitions, one with std. 6T SRAM cell (marked as <8:63>) and other with 9T NOR-CAM cell (marked as <0:7>). The circuit design (In-memory DAC circuit) is proposed by A.K. Ramanathan et. al. [17], which can convert the digital value

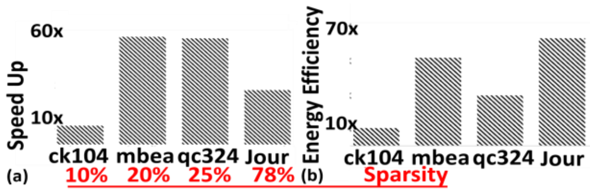

Figure 4: Speed-up and energy comparison of I-NMC against NMC system for various sparse matrix datasets with sparsity information.

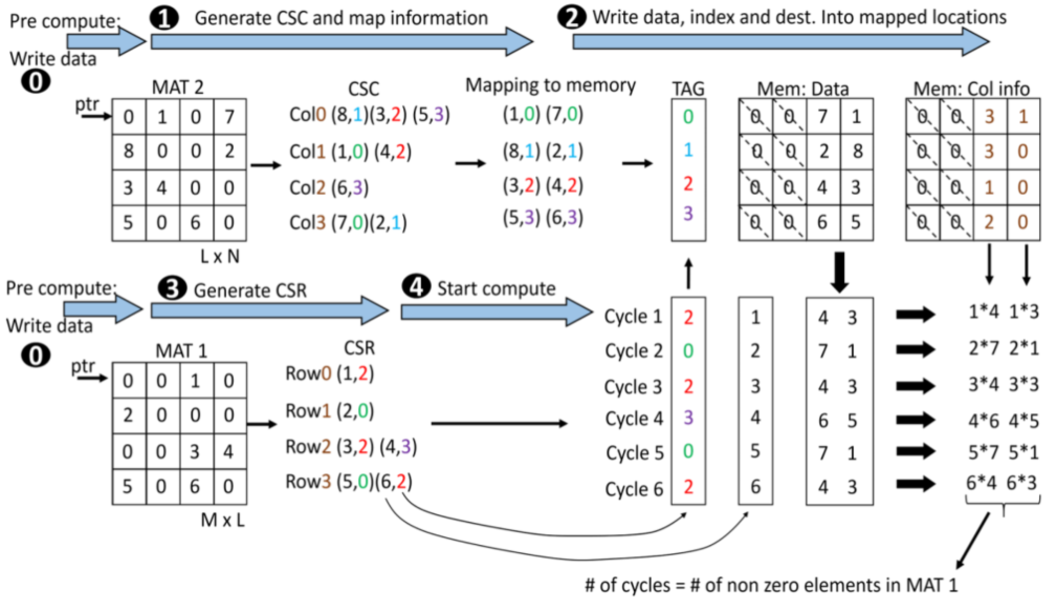

Figure 5: SpGEMM design flow. Bulk comparison is performed through IMC while index handling and MAC operation are designed as NMC hardware.

stored in the SRAM memory cell into its analog equivalent using the CAM peripherals. The in-memory DAC conversion is done in parallel across all the rows, augmenting the I-NMC with massively parallel in-memory computation. In conjunction to the in-memory DAC circuit design, the system requires additional near memory compute logics (registers, I-NMC compute unit, FSM- control logic) to perform sparse matrix index handling mechanism within the sub-array. This system intelligently reduces the number of checks by using compare operations and sends the corresponding data out to the compute logic only on the index match.

The I-NMC sparse matrix index handling system is compared against a NMC baseline which executes similar index handling mechanism for various sparse matrix datasets from SuiteSparse Matrix Collection [18] shown in Figure 4. The I-NMC system shows high speed-up and energy gains mainly attributing to the lesser data traversals and high parallelism within the memory compared to the NMC system.

5. Case study: SpGeMM

5.1 Design overview

SpGeMM is one of the fundamental computations in executing ML tasks. Matrix workloads have become ubiquitous and the fundamental computations in many present-day applications. We observe variable sparsity anywhere from 60% to 99% in these workloads. Several works from the literature [19-21] describe efficient sparsity handling technique. With the increase in matrix dimensions coupled with the sparsity nature, several hardware approaches end up performing poorly. The resultant memory and compute bottleneck can be alleviated by I-NMC design style as it will assist in higher performance as well as avoid un-necessary computation and thereby saving power.

This design relies on four key steps depicted in Figure 5. At least one of the two Matrix must be stored in memory for there to be any potential benefit from I-NMC. Hence as part of STEP 0, Dense input Matrix data is written into the memory. Either both matrix data can be written simultaneously, or second matrix can be a streaming input to the compression hardware. This approach works well for storing the entire weight matrix of initial stages of several Neural Networks. As part of STEP1, Matrix data is parsed row-wise and for every parsed row, non-zero element and the corresponding column information is generated. The rearranged version of the from Figure 5 helps in efficient data mapping. Using the index information, multiple non-zero column elements per row are squeezed together and stored in the corresponding memory locations. The index generation hardware requires only around 1% of the I-NMC entire design area. Index information is then stored in a Content Addressable Memory (CAM) or TAG memory for TAG matching during the Multiplication operation. Nonzero data and the column information are stored in separate SRAM arrays but at the same address location.

This design can extract the non-zero elements and the corresponding index every clock cycle. Thus, the index from column position of one Matrix is matched with all the indices form row information in one clock cycle form the second matrix data. CAM search enable this matching across all the index generated from STEP 1. TAG match results in reading

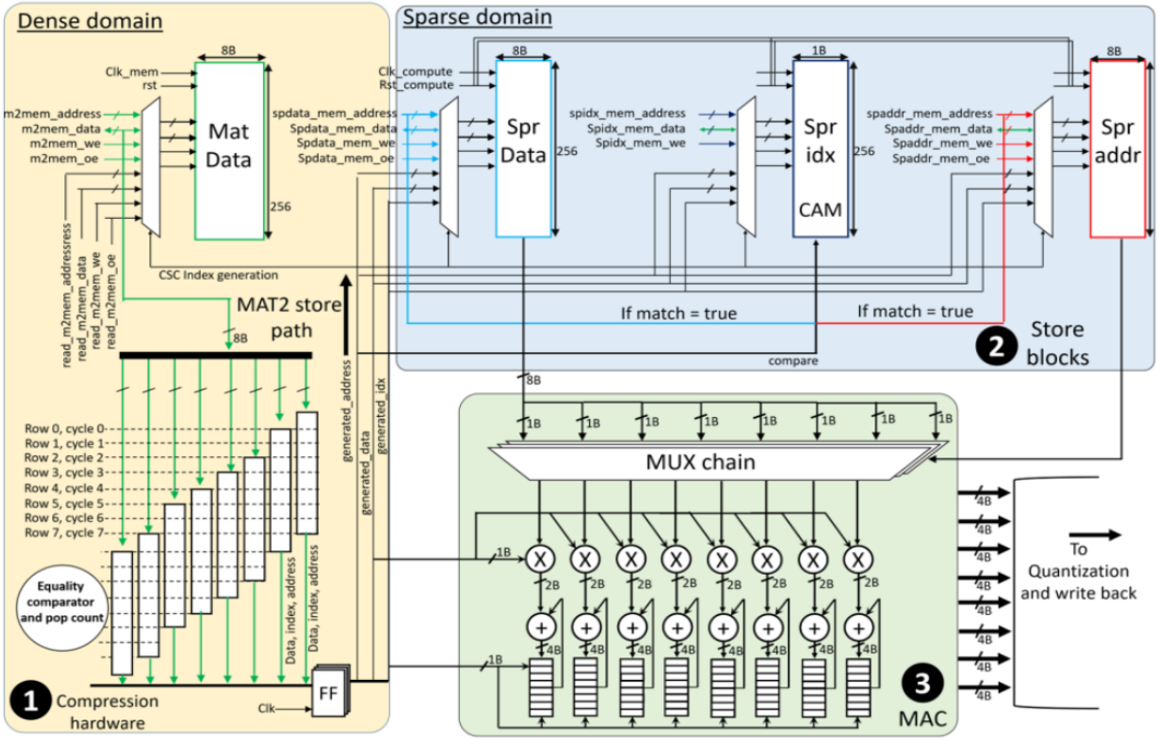

Figure 6: SpGEMM design flow. Bulk comparison is performed through IMC while index handling and MAC operation are designed as NMC hardware.

all non-zero data (from Mem: Data) and destination address (from MEM: Col info) belonging to a column. In this case, CAM operation is equivalent to performing search across entire matrix row and hence can be considered as IMC operation. Elementwise multiplication and accumulation follow this step. Once the dense matrix is represented in a sparse format, we can perform multiplication every clock cycle. Therefore, number of non-zero elements directly determines the computation cycles.

5.2 Microarchitecture

This section describes the Microarchitecture, memory and near memory logic requirements for the design. Figure 6 shows the simplified microarchitecture with the SRAM memory divided into three sub regions and a CAM memory to store the sparse information in the form of indices. In this design, the maximum size of one SRAM array is 256x8B wide. However, the approach remains same for larger matrices occupying more memory space. The design also assumes that each matrix element is 1Byte wide and hence the memory array can hold up to 8 elements of a row. Inputs are stored in "Mat Data" array and the controller configure the read/write control signals, generates address and computes the sparsity information. Comparator and pop count logic to obtain the sparse index, non-zero data and the destination. A wider memory or multiple instances with the same hardware will enable index generation across a wider data set. A single hardware can be used to generate both row-wise and column-wise sparse representations. We can also choose to replicate the hardware to perform these two operations simultaneously. Index and data corresponding to non-zero elements are stored in the address spaces of "SprData" and "SprAddr".

Since the memory array is 8B wide, an eight-stage pipeline hardware generates the first compressed information after 8 cycles of data read and then each clock cycle generates sparse information for the rest of the matrix data. As and when the sparse index information is computed for second matrix, the index is compared for a TAG match. A match means that both the matrices have nonzero elements at the same index locations and should be multiplied. A match fail refers to at least one zero element in either of the matrices and resultant of multiplication is a 'zero" and hence we can avoid the computation on those rows completely.

Chiplet based integration approach can be useful in separating the Memory and logic portions of the design. 3D integration with logic on memory stacking further helps in reducing the overall memory footprint without compromising on the performance.

5.3 Computation cycle for large matrix dimension

Overall computation cycle for N x M Matrix is shown in TABLE I. For large matrices and when the second matrix sparse data is already stored in SRAM, number of non-zero elements directly translate to the overall computation cycle. Compression can be parallelized across the columns and

TABLE I: Overall SpGeMM compute cycle

	Operation (NxM Matrix 1)	Cycles
Step 0	Data write phase to SRAM	N
Step 1	Row read + index generation	1 + 8
Step 2	IMC compare + retrieve destination location	1 + 1
Step 3	Multiplication operation	1
Step 4	Accumulate + store	1 + 1
Total	N rows of a matrix	~N+ Δ

hence saves much computation latency. Near memory accelerator minimizes the data movement cost by not repeatedly accessing the storing the data. Compared to this design SpGEMM computation by making use of SpVM techniques require N x M cycles without compressing one of the matrices.

6. Conclusion

Both IMC and NMC are very promising techniques to overcome the performance bottleneck due to increased data traffic while executing ML tasks. While SRAM based IMC, techniques are most suited for Boolean operations, CAM based IMC amalgamated with NMC techniques show further opportunities in accelerating fundamental computations of AI and ML tasks. Throughout this paper, we described qualitatively with examples that show benefits of IMC using PIM_OPC and advantages of combining IMC and NMC by taking a specific example of SpGeMM. These techniques can be further extended for several graph analytics tasks. Combination of different IMC and NMC techniques can further unlock opportunities in accelerating applications and tasks beyond those which are described in this work.

7. References

[1] George, Sumitha, et al. "MDACache: Caching for multi-dimensional-access memories." 2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). IEEE, 2018.

[2] S. R. Swamy Saranam et al., "Optimization of Intercache Traffic Entanglement in Tagless Caches With Tiling Opportunities," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 39, no. 11, pp. 3881-3892, Nov. 2020

[3] S. K. Thirumala et al , "Non-Volatile Memory utilizing Reconfigurable Ferroelectric Transistors to enable Differential Read and Energy-Efficient In-Memory Computation," 2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED), Lausanne, Switzerland, 2019, pp. 1-6

[4] Parveen, F et al., Hielm: Highly flexible in-memory computing using stt mram. In 2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)

[5] S. Kim et al., "Processing-in-memory in High Bandwidth Memory (PIM-HBM) Architecture with Energy-efficient and Low Latency Channels for High Bandwidth System," IEEE 28th EPEPS.

[6] Srinivasa, Srivatsa, et al. "Monolithic 3D+-IC based reconfigurable compute-in-memory SRAM Macro." 2019 Symposium on VLSI Technology. IEEE, 2019.

[7] Ramanathan et al. "Look-up table based energy efficient processing in cache support for neural network acceleration." 2020 53rd Annual IEEE/ACM (MICRO).

[8] Fu-Kuo Hsueh et al., "TSV-free FinFET-based monolithic 3D+-IC with computing-in-memory SRAM cell for intelligent IoT devices," in 2017 IEDM.

[9] S.Aga et al., "Compute Caches," IEEE International Symposium on High Performance Computer Architecture (HPCA), Austin, TX, 2017

[10] Srinivasa, Srivatsa, et al. "Monolithic-3D Integration Augmented Design Techniques for Computing in SRAMs." 2019 IEEE ISCAS.

[11] Eckert, Charles, et al. "Neural cache: Bit-serial in-cache acceleration of deep neural networks." 2018 ACM/IEEE 45th ISCA. IEEE, 2018.

[12] D. Fujiki et al., "Duality Cache for Data Parallel Acceleration," 2019 ACM/IEEE 46th ISCA 2019.

[13] Jintao Zhang et al., "In-memory computation of a machine learning classifier in a standard 6T SRAM array," IEEE Journal of Solid-State Circuits, 2017

[14] Chen et al., Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," in IEEE JSSC, pp. 127-138, 2017.

[15] Y. Chen et al., A machine-learning supercomputer. In 47th Annual IEEE/ACM International Symposium on Microarchitecture, pages 609–622, Dec 2014.

[16] Yakun Sophia Shao et al., Simba: Scaling deep-learning inference with multi-chip-module-based architecture. In Proceedings of MICRO ’52, 2019.

[17] A.K. Ramanathan, et al., “Monolithic 3D+-IC Based Massively Parallel Compute-in-Memory Macro for Accelerating Database and Machine Learning Primitives” in 2020 IEDM.

[18] https://sparse.tamu.edu/

[19] Kanellopoulos et al., SMASH: Co-designing software compression and hardware-accelerated indexing for efficient sparse matrix operations 2019 Microarchitecture, MICRO, pp. 600-614.

[20] W. Liu and B. Vinter, "An Efficient GPU General Sparse Matrix-Matrix Multiplication for Irregular Data," 2014 IEEE 28th International Parallel and Distributed Processing Symposium, Phoenix, AZ, 2014, pp. 370-381, doi: 10.1109/IPDPS.2014.47.

[21] A. K. Ku, J. Y. Kuo and J. Xue, "Hardware Support for Efficient Sparse Matrix Vector Multiplication," 2008 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing, Shanghai, 2008, pp. 37-43, doi: 10.1109/EUC.2008.154.