Computing-in-Memory with Spintronics

Shubham Jain[1], Sachin Sapatnekar[2], Jian-Ping Wang[2], Kaushik Roy[1], Anand Raghunathan[1]
[1]School of Electrical and Computer Engineering, Purdue University
[2]Department of Electrical and Computer Engineering, University of Minnesota
[1]{jain130,kaushik,raghunathan}@purdue.edu, [2]{sachin,jpwang}@umn.edu

***Abstract*—In-memory computing is a promising approach to alleviating the processor-memory data transfer bottleneck in computing systems. While spintronics has attracted great interest as a non-volatile memory technology, recent work has shown that its unique properties can also enable in-memory computing. We summarize efforts in this direction, and describe three different designs that enhance STT-MRAM to perform logic, arithmetic, and vector operations and evaluate transcendental functions within memory arrays.**

I. Introduction

Modern computing workloads such as data analytics, machine learning, graphics, and bioinformatics operate on large datasets, leading to frequent accesses to off-chip memory. Consequently, a significant amount of time and energy is spent in the movement of data between the processor and memory, causing a major bottleneck in computing systems.

Among various approaches to address the processor-memory bottleneck, the closer integration of logic and memory has garnered significant interest. These approaches are variedly referred as near-memory computing, processing-near-memory, logic-in-memory, computing-in-memory, and processing-in-memory. These efforts may be classified into two broad categories based on the degree of integration between logic and memory. In *Near-Memory Computing* [1]–[7], logic is moved closer to memory, *e.g.*, within the same package. On the other hand, *In-Memory Computing* [8]–[21] refers to performing computations within a memory array itself.

Although both near- and in-memory computing alleviate the processor-memory bottleneck, the latter fundamentally blurs the distinction between computation and memory. Additionally, it has the benefit of reducing the number of memory accesses by moving computations to the memory rather than bringing data to the processor. Therefore, in-memory computing has drawn great interest in recent years [8]–[21].

Spintronics has emerged as a promising candidate for future memories due to its desirable attributes such as high density, non-volatility, near-zero leakage, high endurance, low voltage read and write operations, and compatibility with the CMOS manufacturing process. In particular, Spin Transfer Torque Magnetic RAM (STT-MRAM) has attracted great interest with prototype demonstrations and commercial offerings [22], [23]. In this paper, we present an overview of approaches to in-memory computing with STT-MRAM. We describe three different in-memory computing designs – STT-CiM [12], ROM-Embedded MRAM [13], and CRAM [10] that propose modifications to the peripherals or the bit-cells of STT-MRAM to enable in-memory computing. We show that by leveraging the unique attributes of spintronic memories, these designs perform computations (logic operations, scalar and vector arithmetic, and transcendental functions) within the memory array.

STT-CiM [12] is an example of in-memory computing with spintronic memory that proposes modifications to the peripherals while retaining the core bit-cell and array structure of STT-MRAM intact. It exploits the resistive nature of spintronic memories to simultaneously enable multiple wordlines in an STT-MRAM array, leading to multiple bit-cells being connected to each bit-line. With modifications to the peripheral circuitry, STT-CiM senses the effective resistance of each bit-line to directly compute functions of multiple words. Based on this principle, STT-CiM enables logic, arithmetic, and vector computations to be performed in-memory.

In contrast, ROM-Embedded MRAM [13] and CRAM [10] take a different approach to realize computations in STT-MRAM. They propose modifications to both the bit-cell and the peripherals to enable in-memory operations.

ROM-Embedded MRAM [13] introduces a second bit-line, and during the design process selectively connects the memory element [*i.e*, Magnetic Tunnel Junction (MTJ)] to one of the two bit-lines to store 1-bit of read-only data in each bit-cell. The STT-MRAM can be used in two (read-only and regular) modes. The read-only mode can be used to store look-up tables and used to evaluate complex mathematical functions such as transcendental functions. On the other hand, CRAM [10] introduces an additional transistor in each bit-cell (2T-1R) to realize more complex operations in the STT-MRAM. It allows the results of computation to be written back within the memory array itself, without the need to take data to the array periphery. CRAM provides a reconfigurable platform that can be used to construct adders, multipliers, dot product computations, *etc.*, in the memory array.

The rest of the paper is organised as follows. Section II provides a taxonomy of approches to computing-in-memory. Section III provides the necessary background on STT-MRAM. Section IV describes approaches to in-memory computing with spintronics. Section V discusses challenges and opportunities, and concludes the paper.

This work was supported in part by C-SPIN, one of the six centers of STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA and in part by C-BRIC, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA.

Near-Memory Computing (Placing logic near memory)		In-Memory Computing (Moving computations within memory array itself)	
Application-specific	General-purpose	Application-specific	General-purpose
Interpolation memory [4], Neurocube [7]	3D stacking (HMC, HBM) [5,6], IRAM [1]	Automata Processor [8], Compute-Memory [9], In-Memory Classifier [18], RENO [19], Spindle [20], PRIME [21], ROM-Embedded MRAM [13]	Pinatubo [11], bulk bitwise in-DRAM [16] STT-CiM [12], CRAM [10], MAGIC [14]

Fig. 1. Taxonomy of Approaches to Near- and In-Memory Computing

II. Taxonomy of Computing-in-memory

The closer integration of logic and memory is variedly referred to as near-memory computing, processing-near-memory, processing-in-memory, logic-in-memory, and computing-in-memory. Table 1 presents a taxonomy of these efforts based on the degree of integration (Near- or In-memory), the scope of use (application-specific or general-purpose).

Near-memory Computing. Near-memory computing refers to bringing logic or processing elements closer to memory. However, the memory and computation units are still distinct. Several efforts have explored near-memory computation at various levels of the memory hierarchy [1]–[7]. Intelligent-RAM [1], which integrates a processor and DRAM in the same chip to increase processor-to-memory bandwidth, is an early example of near-memory computing. Other early examples include active pages [2] and active disks [3], which enable computations within main memory and secondary storage, respectively. Interpolation memory [4], which combines look-up tables stored in memory with interpolation logic to evaluate complex mathematical functions, and Neurocube [7], which proposes a neural network accelerator based on 3D integration of processors and memory, are examples of application-specific near-memory computing. In recent years, with the advent of advanced packaging technology and 3D integration, near-memory computing has gained considerable momentum with industrial efforts like Hybrid Memory Cube (HMC) [5] and High Bandwidth Memory (HBM) [6].

In-memory Computing. In-memory computing [8]–[21] fundamentally blurs the boundary between computation and memory, since the memory array itself behaves like a computing unit. Instructions to perform computation are sent to memory instead of bringing data from memory to the processor. In-memory computing efforts can be further classified based on whether they target application-specific or general-purpose computations. Examples of application-specific in-memory computing include vector-matrix multiplications [18]–[21] and sum-of-absolute difference computations [9]. ROM-embedded MRAM [13] and Micron's automata processor [8] can also be viewed as examples of in-memory computing that target specific operations such as pattern matching or the evaluation of transcendental mathematical functions. General-purpose in-memory computing designs focus on a broader class of operations. Many proposals within this category restrict themselves to bit-wise Boolean logic operations (*e.g.*, AND/NAND/OR/NOR/XOR) [11], [14]–[17]. In contrast, STT-CiM [12] and CRAM [10] can realize a broader range of logic, arithmetic, and vector operations.

III. Preliminaries

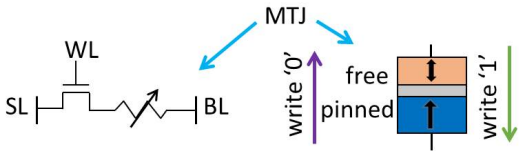

Fig. 2. STT-MRAM bit-cell

An STT-MRAM bit-cell comprises of an access transistor in series with a Magnetic Tunneling Junction (MTJ) as shown in Figure 2. An MTJ is a spintronic device consisting of a pinned layer with a fixed magnetic orientation, and a free layer whose magnetic orientation can be switched, separated by a tunneling oxide. An MTJ behaves likes a programmable resistor whose resistance is determined by the relative magnetic orientation of the free and the pinned layers. There are two stable magnetic orientations – parallel and anti-parallel – that lead to two distinct resistances R_P and R_{AP}, respectively. The two resistance states can encode 1-bit of information, *e.g.*, we may assume R_P to represent logic '1' and R_{AP} to represent logic '0'. A read operation is performed by enabling the wordline (WL) and applying a voltage (V_{read}) across the bit-line (BL) and the source-line (SL). The resultant current (I_P or I_{AP}) flowing through the bit-cell is compared against a global reference to determine the value stored in the bit-cell. A write operation is performed by passing a current greater than the critical switching current of the MTJ. The direction of the write current determines the value written into the cell, as shown in Figure 2. Write operations in STT-MRAM are stochastic in nature, and the write failure rate is determined by the magnitude and duration of the write current. STT-MRAM also suffers from read decision failures, where the values stored in the bit-cells are incorrectly sensed under process variations, and read disturb failures, where a read operation corrupts the value stored in the bit-cell. In order to mitigate these failures, a range of techniques from device and circuit optimizations to error correcting codes [24]–[26] have been proposed.

IV. In-Memory Computing with Spintronics

In this section, we describe three different designs — STT-CiM, ROM-embedded MRAM and CRAM – that enable in-

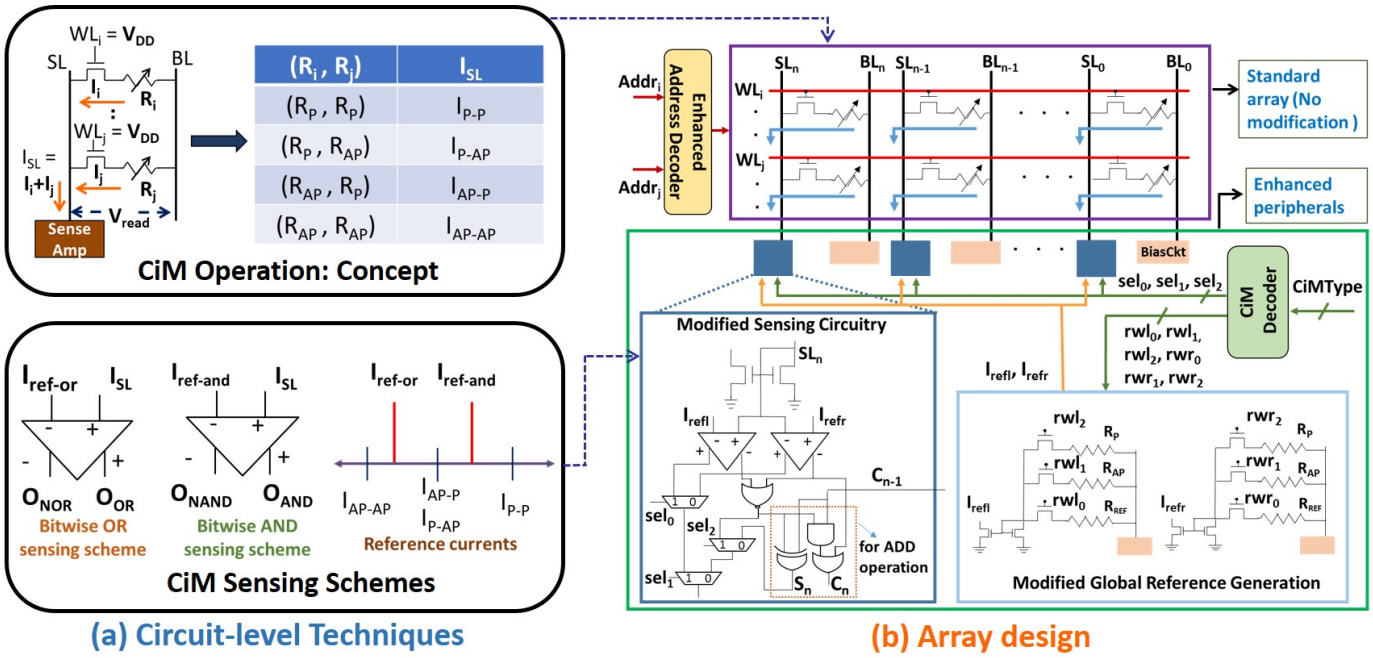

Fig. 3. STT-CiM Overview

memory computing with spintronics.

A. STT-MRAM based Compute-in-Memory (STT-CiM)

The key idea behind STT-CIM [12] is to enable multiple wordlines simultaneously, and directly compute logic functions of the enabled words. STT-CiM does not modify the core array structure or the bit-cell. By suitably enhancing the peripheral (address decoding, sensing and reference generation) circuitry, it is possible to perform Compute-in-memory (CiM) operations on words stored in the STT-MRAM array. We illustrate the concept behind STT-CiM using Figure 3(a). As shown, CiM operations are performed by activating two wordlines (WL_i and WL_j) simultaneously, and applying a read voltage (V_{read}) at the bit-line (BL). The resultant current (I_{SL}) flowing through the source-line (SL) is a summation of the currents flowing through the two bit-cells, and therefore is a function of the bit-cells' logic states, as shown in Figure 3(a). STT-CiM proposes sensing mechanisms to distinguish between these values and thereby compute logic functions of the stored words.

Figure 3(a) illustrates the sensing schemes utilized to realize various bit-wise operations. In order to realize bit-wise OR and NOR operations, (I_{SL}) is connected to the positive terminal of the sense amplifier and the reference current (I_{ref-or}) is fed to the negative input. Choosing I_{ref-or} to be between I_{AP-AP} and I_{AP-P} yields the OR/NOR operation at the positive/negative output of the sense amplifier. Similarly, the bit-wise AND sensing scheme is used for realizing bit-wise AND/NAND operations. A bit-wise XOR operation is realized by combining both the sensing schemes shown in Figure 3(a), and O_{AND} and O_{NOR} are fed to a CMOS NOR gate to obtain O_{XOR}. In other words, $O_{XOR} = O_{AND}$ NOR O_{NOR}.

An ADD operation is realized in STT-CiM by leveraging the ability to concurrently perform multiple bit-wise operations using the sensing mechanisms described above. Figure 4 illustrates an ADD operation performed on the n^{th} bit of words A and B that are stored in the same column of the memory array. As shown, the sum (S_n) and carry (C_n) of the n^{th} stage are computed using bit-wise operations A_n XOR B_n and A_n AND B_n, and the carry-in from the previous stage (C_{n-1}). These bit-wise operations can be realized in-memory. Note that the sensing schemes described above allow STT-CiM to compute bitwise XOR and bitwise AND simultaneously, thereby enabling ADD operations with only a single array access.

Full-adder function	$S_n = (A_n \text{ XOR } B_n) \text{ XOR } C_{n-1}$ (bitwise operation) $C_n = ((A_n \text{ XOR } B_n) \text{ AND } C_{n-1}) \text{ OR } (A_n \text{ AND } B_n)$
In-Memory ADD	$S_n = O_{XOR} \text{ XOR } C_{n-1}$ $C_n = (O_{XOR} \text{ AND } C_{n-1}) \text{ OR } O_{AND}$

Fig. 4. STT-CiM ADD operation [12]

Next, we present the STT-CiM array design that leverages the circuit-level techniques described above. Figure 3(b) shows the STT-CiM array that is comprised of the standard STT-MRAM array with modified peripherals. It has an additional input (CiMType) to indicate the type of CiM operation (if any) that needs to be performed during a memory access. The CiM decoder uses CimType to generate the required control signals to the memory array and peripheral circuits. Since a CiM operation requires the activation of two wordlines, STT-CiM has a modified row address decoder that takes two addresses and can activate any two wordlines in the memory array. The write peripheral circuits remain same as in STT-MRAM, since write operations are unchanged. Figure 3(b) shows the modified sensing and reference generation circuitry.

The sensing circuitry consists of 2 sense amplifiers, a NOR gate, 3 multiplexers, and 3 additional logic gates to enable in-memory ADD operations. The reference generation circuitry includes two reference stacks, one for each of the two sense amplifiers. Each stack comprises of three hard-wired bit-cells programmed to offer resistances R_P, R_{AP}, and R_{REF}(where $R_P > R_{REF} > R_{AP}$). As illustrated in Figure 3(b), the CiM decoder generates the control signals to the sensing and reference generation circuitry based on the CiMType to realize various compute-in-memory operations.

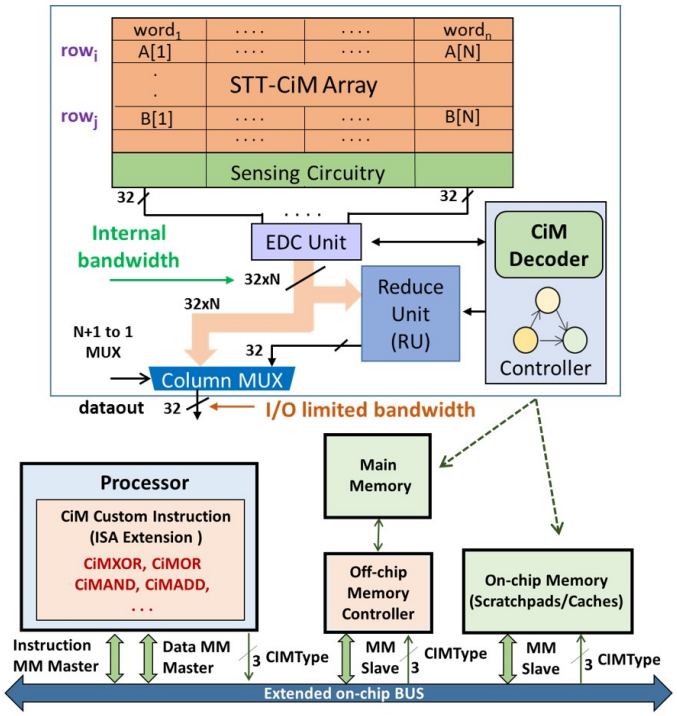

Fig. 5. STT-CiM: System-Level Integration

In order to further improve the efficiency of in-memory computing for data parallel workloads, STT-CiM exploits the high internal bandwidth of memory to perform CiM operations on all elements of a vector concurrently. However, the output of CiM operations performed on vectors is also a vector, which cannot be sent back to the processor in a single access. To resolve this, STT-CiM leverages the observation that vector operations are often followed by reduction operations. Therefore, STT-CiM contains a Reduce Unit, which is used to perform reductions on the computed vector to produce a scalar result, and the resultant scalar is sent back to the processor, as shown in Figure 5.

STT-CiM can be easily integrated in a programmable processor based system with enhancements to the processor's instruction-set architecture (ISA) and on-chip bus. Moreover, it is a general design that can be deployed within both on-chip (scratchpad or cache) memories and off-chip memory, as shown in Figure 5.

B. ROM-Embedded MRAM

ROM-Embedded MRAM (R-MRAM) [13] takes a different approach to realize in-memory computing with STT-MRAM. It proposes a simple modification to STT-MRAM bit-cells to equip them with the ability to concurrently store both 1-bit of read-only and 1-bit of programmable (read/write) data. Specifically, it introduces an additional bit-line in each bit-cell and selectively connects the MTJ to one of the two bit-lines at design time. This choice can be used to store 1-bit of read-only data. The programmable bit is still stored in the MTJ. Figure 6 illustrates the R-MRAM bit-cell that has two bit-lines (BL0 and BL1) in contrast to a standard STT-MRAM bit-cell (described in section III). As shown, during design time, the read-only data is stored in the bit-cells by selectively masking the VIA below the bit-lines, thereby connecting the MTJ to the first or second bit-line. A bit-cell with its MTJ connected to BL0 stores the read-only logic state '0', whereas a bit-cell whose MTJ is connected to BL1 represents read-only logic '1'. The R-MRAM's ability to store additional read-only data in every bit-cell presents possibilities of computing transcendental functions (sine, logarithm, sigmoidal, gamma correction, *etc.*) in-memory by implementing them as look-up tables stored in the read-only memory (ROM) bits. We next describe the peripheral circuit enhancements and sensing schemes for R-MRAM that allow seamless access to both read-only and programmable data.

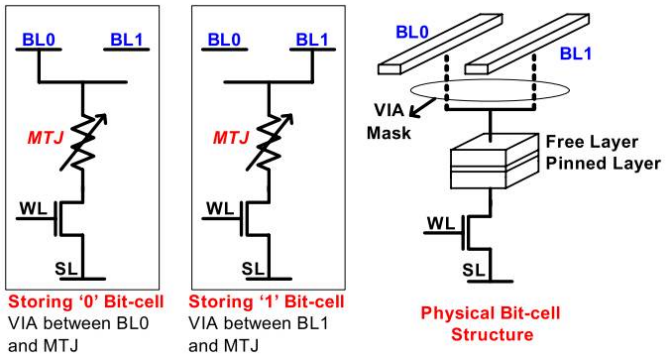

Fig. 6. R-MRAM bit-cell design [13]

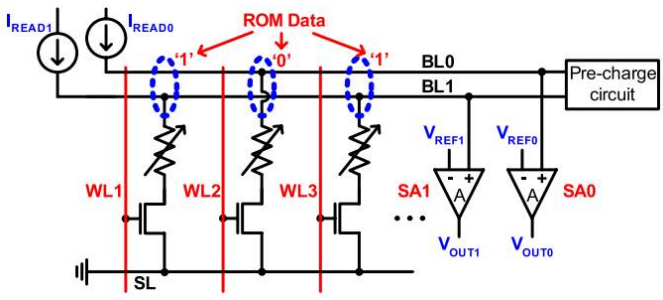

Fig. 7. R-MRAM peripheral design [13]

Figure 7 illustrates the read peripheral circuitry of R-MRAM, which is comprised of two sense amplifiers (SA0 and SA1), a pre-charge circuit, and two current sources (I_{READ0} and I_{READ1}). The positive terminals of the sense amplifiers SA0 and SA1 are connected to the bit-lines BL0 and BL1, respectively, and their negative inputs are connected to the reference voltages (V_{REF0} and V_{REF1}). We first describe the regular mode operation in which the peripherals are configured

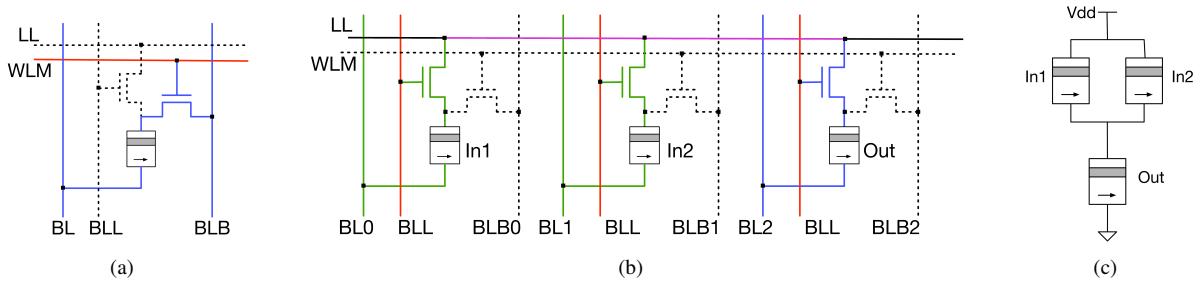

Fig. 8. (a) CRAM cell in (default) memory mode. (b) 2-input CRAM gate in logic mode. An implementation of (c) 2-input NAND, using MTJs.

to read the programmable bit stored in the MTJ. In this mode, the bit-lines (BL0 and BL1) are pre-charged to a reference voltage ($V_{REF} = V_{REF0} = V_{REF1}$), followed by the activation of a wordline and the current sources (I_{READ0} and I_{READ1}). The resultant current flowing through the bit-cell originates from either I_{READ0} or I_{READ1} (as only one bit-line is connected) and goes through the corresponding bit-line (BL0 or BL1) to the source-line (SL). The final voltage of the connected bit-line (BL0 or BL1) is dependent on the logic state of the MTJ, and can take a value less than V_{REF} for R_P or greater than V_{REF} for R_{AP}. Consequently, causing the associated sense amplifier output (V_{OUT0} or V_{OUT1}) to read a value complementary to the MTJ's logic state. The bit-line (BL0 or BL1) that is not connected to the activated bit-cell remains coupled only to the current source, and thereby witnesses a surge in its voltage due to the charging of its capacitance, causing the coupled sense amplifier's output (V_{OUT0} or V_{OUT1}) to be logic '1'. Performing a NAND operation on the sense amplifiers' outputs yields the stored logic value. In other words, the programmable bit is evaluated as V_{OUT0} NAND V_{OUT1}.

Next, we describe the ROM-mode read operation in which the peripherals are configured to read the stored read-only memory (ROM) bits. In this mode, bit-lines are pre-charged to 0V followed by activation of the wordline and the current sources. The bit-line (BL0 or BL1) that is not connected to the source-line (SL) causes the coupled sense amplifier output to be logic '1', as described above. In contrast, the connected bit-line causes the associated sense amplifier output to be logic '0' due to the pre-charging of the bit-lines to 0V. Thus, SA0 senses the stored read-only data, whereas SA1 yields its complementary value.

C. Computational RAM (CRAM)

The Computational RAM (CRAM) platform [10] uses a 2T-1MTJ bit cell, adding another transistor to the traditional 1T-1MTJ STT-MRAM cell (Fig. 8(a)). The array can either function as a standard STT-MRAM memory, or can allow *in situ* computation of logic functions within the memory, without requiring data to be taken to the periphery. In **memory mode**, WLM is logic high, and the cell behaves like a 1T-1R MRAM cell, using BL and BLB for read/write operations.

We illustrate **logic mode** by showing the operation of a two-input CRAM gate in Fig. 8(b), where the first two cells contain the input data and the third receives the output. For these cells, BLL is enabled, which connects them to the shared logic line (LL), and a voltage pulse is applied to BL2 while BL[0:1] are grounded. The three MTJs effectively create the circuit in Fig. 8(c) [27], communicating through LL. The states of the input MTJs and the preset state of the output MTJ determine the current through the output MTJ. If this current is large enough, the output MTJ switches; otherwise it retains its preset state.

By altering the configuration of inputs, the supply voltage, and the preset state, a variety of logic functions (NOT, BUFFER, (N)AND, (N)OR, MAJ, $\overline{\text{MAJ}}$) can be realized. Since several types of universal gates can be realized, any logic function can be built in the CRAM. It is easy to use these functions to construct adders, multipliers, dot product computations, *etc.*, in the CRAM.

The resilience of CRAM computations depends on the difference in resistance between the two states, i.e., on TMR $= (R_{AP} - R_P)/R_P$; generally speaking, higher TMR values are better. Current mainstream technologies show TMR values of about 100-150%, although values of over 600% have been shown in experiments, and roadmaps predict a technology path towards TMR values of over 1000% [28].

Thus, CRAM supports *true in-memory* computing by reconfiguring cells as logic gates *within* the memory array. Each MTJ can serve as an input or as an output of a logic gate, as the computational demands of the workload evolve over time. Further, because all cells in the array are identical, the order of applied voltage pulses (as opposed to the physical layout) determines inputs and outputs. This allows great flexibility in reconfiguring the CRAM array as memory or as various types of logic gates. The combination of logic reconfigurability and the movement of data during computation raises interesting problems in the domain of data placement and computation scheduling on the CRAM.

Preliminary analysis [29] indicates that CRAM can outperform CMOS solutions in energy efficiency with competitive throughput: CRAM can deliver higher throughput at iso-energy, or lower energy consumption at iso-throughput. The speedup of CRAM arises from several factors. First, each row can perform computations independently, allowing large levels of parallelism and making CRAM an ideal candidate for data-intensive emerging applications. Second, logic computations are performed with subarrays of the memory and incur low communication latencies. In contrast, approaches that take data to the edge of memory incur larger communication latencies

as well as serial bottlenecks.

V. Conclusion

In-memory computing is a promising approach to addressing the processor-memory bottleneck. In this paper, we presented an overview of various approaches to in-memory computing with spintronics. We discussed three designs that leverage the unique attributes of spintronics to realize a range of logic, arithmetic, vector and transcendental functions in STT-MRAM. While the results from these studies are promising, several challenges remain to be overcome. Most previous work has focused on the design of the circuit building blocks, leaving open architectural questions such as how memories with computational capabilities are best integrated into a computing system. For example, at what level(s) of the memory hierarchy should in-memory computation be used? Ensuring sufficient sensing margins for conventional read and write operations in STT-MRAM under variations is challenging, and this challenge is further exacerbated when sensing a combination of values from multiple bit-cells. How can we compute in-memory reliably under process variations? Finally, what programming abstractions and software tools need to be developed so that in-memory computing can be used with minimal programmer effort? These challenges need to be addressed in order to realize the potential of in-memory computing.

References

[1] D. Patterson et al. Intelligent ram (IRAM): Chips that remember and compute. In *Proc. ISSCC., 1997*.

[2] M. Oskin et al. Active Pages: A computation model for intelligent memory. In *Proc. ISCA*, Jun 1998.

[3] E Riedel et al. Active disks for large-scale data processing. *Computer*, 34(6):68–74, 2001.

[4] Q Zhu et al. Design automation framework for application-specific logic-in-memory blocks. In *Proc. ASAP*, July 2012.

[5] J. T. Pawlowski. Hybrid memory cube (HMC). In *Hot Chips*, volume 23, 2011.

[6] D. Lee et al. 25.2 A 1.2 V 8Gb 8-channel 128GB/s high-bandwidth memory (HBM) stacked DRAM with effective microbump I/O test methods using 29nm process and TSV. In *Proc. ISSCC*, pages 432–433. IEEE, 2014.

[7] D. Kim et al. Neurocube: A Programmable Digital Neuromorphic Architecture with High-Density 3D Memory. In *Proc. ISCA*, June 2016.

[8] P. Dlugosch et al. An efficient and scalable semiconductor architecture for parallel automata processing.

[9] M Kang et al. An energy-efficient VLSI architecture for pattern recognition via deep embedding of computation in SRAM. In *Proc. ICASSP*, May 2014.

[10] J.-P. Wang and J. Harms. General structure for computational random access memory (CRAM), 2015. US Patent 9224447 B2.

[11] S Li et al. Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories. In *Proc. DAC*, June, 2016.

[12] S Jain et al. Computing in Memory with Spin-Transfer Torque Magnetic RAM. *CoRR*, abs/1703.02118, 2017.

[13] D. Lee et al. R-MRAM: A ROM-Embedded STT MRAM Cache. *IEEE EDL*, Oct 2013.

[14] N. Talati et al. Logic Design Within Memristive Memories Using Memristor-Aided loGIC (MAGIC). *Trans. TNANO*, pages 635–650, July 2016.

[15] J Reuben et al. Memristive Logic:A framework for evaluation and comparison. In *Proc. PATMOS*, SEP, 2017.

[16] V. Seshadri et al. Fast Bulk Bitwise AND and OR in DRAM. *Trans. CAL*, July 2015.

[17] W. Kang et al. In-Memory Processing Paradigm for Bitwise Logic Operations in STT-MRAM. *IEEE Tran. on Magnetics*, 2017.

[18] J. Zhang et al. A machine-learning classifier implemented in a standard 6T SRAM array. In *Proc. VLSI-Circuits*, pages 1–2, June 2016.

[19] X. Liu et al. RENO: A high-efficient reconfigurable neuromorphic computing accelerator design. In *Proc. DAC*, June 2015.

[20] S. G. Ramasubramanian et al. SPINDLE: SPINtronic Deep Learning Engine for large-scale neuromorphic computing. In *Proc. ISLPED*, Aug 2014.

[21] P. Chi et al. PRIME: A Novel Processing-in-memory Architecture for Neural Network Computation in ReRAM-based Main Memory. In *Proc. ISCA*, 2016.

[22] http://www.everspin.com/ .

[23] D. Apalkov et al. Spin-transfer Torque Magnetic Random Access Memory (STT-MRAM). *J. Emerg. Technol. Comput. Syst.*, May 2013.

[24] K. W. Kwon et al. High-Density and Robust STT-MRAM Array Through Device/Circuit/Architecture Interactions. *Nanotechnology, IEEE Trans.*, Nov 2015.

[25] B. D. Bel et al. Improving STT-MRAM density through multibit error correction. In *Proc. DATE*, March 2014.

[26] W. Kang et al. A low-cost built-in error correction circuit design for STT-MRAM reliability improvement. *Microelectronics Reliability*, 2013.

[27] A. Lyle et al. Direct communication between magnetic tunnel junctions for nonvolatile logic fanout architecture. *Applied Physics Letters*, 97(152504), 2010.

[28] A. Hirohata et al. Roadmap for emerging materials for spintronic device applications. *IEEE Transactions on Magnetics*, October 2015.

[29] Z. Chowdhury et al. Efficient in-memory processing using spintronics. *IEEE Computer Architecture Letters*, 2017.