AM^4: MRAM Crossbar Based CAM/TCAM/ACAM/AP for In-Memory Computing

Esteban Garzón, *Member, IEEE*, Marco Lanuzza, *Senior Member, IEEE*, Adam Teman, *Member, IEEE*, and Leonid Yavits, *Member, IEEE*

***Abstract*— In-memory computing seeks to minimize data movement and alleviate the memory wall by computing *in-situ*, in the same place that the data is located. One of the key emerging technologies that promises to enable such computing-in-memory is spin-transfer torque magnetic tunnel junction (STT-MTJ). This paper proposes AM^4, a combined STT-MTJ-based Content Addressable Memory (CAM), Ternary CAM (TCAM), approximate matching (similarity search) CAM (ACAM), and in-memory Associative Processor (AP) design, inspired by the recently announced Samsung MRAM crossbar. We demonstrate and evaluate the performance and energy-efficiency of the AM^4-based AP using a variety of data intensive workloads. We show that an AM^4-based AP outperforms state-of-the-art solutions both in performance (with the average speedup of about 10 ×) and energy-efficiency (by about 60 × on average).**

***Index Terms*— Non-von Neumann computer architecture, associative processor, associative memories, MRAM, MTJ, double-barrier MTJ, CAM, TCAM, emerging memories.**

I. Introduction

COMPUTING is increasingly dominated by the transfer of large data volumes through bandwidth-limited interfaces to the locations where computations are performed, which hampers the performance and energy-efficiency of conventional computer architectures [1]. This is leading to a change in computing paradigms: instead of moving the data to the computation, the computation is moved closer to the data. The straightforward approach, known as "near-memory computing", places processing units close to memory arrays [2], [3], [4], [5]. An alternative way to overcome the so-called "memory wall" is to compute in-data, i.e., directly within the memory arrays. This paradigm, which we refer to throughout this work as "in-memory computing", employs the same memory cells for both data storage and data processing.

This work is inspired by the 2-transistor 2-resistor (2T2R) magnetoresistive (MRAM) crossbar design, recently unveiled by Samsung [6]. We keep the original topology of Samsung's MRAM crossbar to develop a massively parallel general-purpose in-memory computer architecture. We accomplish this goal by:

1) Converting the crossbar into an associative memory (AM), and
2) Transforming the associative memory into a massively parallel in-memory associative computer.

Specifically, we propose AM^4, a novel associative in-memory computing architecture, and investigate its design trade-offs, explore its design space, and evaluate its performance and energy-efficiency. Additional use cases for AM^4 include Content Addressable Memory (CAM), Ternary CAM (TCAM), and approximate match (similarity search) associative memory (ACAM).

To our knowledge, AM^4 is the first magnetoresistive NAND CAM based on an MRAM crossbar. In a conventional NOR-type CAM [7], [8], [9], [10], the matchline discharges on a mismatch. In a typical CAM and TCAM application, only one memory row matches, while the rest mismatch. Since mismatches are much more frequent than matches, all matchlines need to be precharged prior to every search/lookup, resulting in a very significant energy consumption. In contrast, in a NAND-type CAM, only the matching row(s) discharge, reducing the energy consumption of search/lookup by orders of magnitude.

Our research methodology spans from high-level software simulation through to accurate transistor-level circuit simulation. Our circuit design utilizes a 28 nm FDSOI technology node along with a Verilog-A based compact model for the double-barrier magnetic-tunnel junction (DMTJ) device [11]. Evaluations under exhaustive Monte Carlo simulations show that AM^4 offers a compare time of about 1.4 ns, which consumes and 1.73 fJ of energy. These results are achieved with a bit cell footprint of just 0.138 μm^2. Smith-Waterman optimal sequence alignment algorithm [12] is used to evaluate AM^4 and compare it to state-of-the-art computing in-memory and conventional solutions. Results show that AM^4 can provide orders-of-magnitude performance and energy-efficiency improvement versus traditional von Neumann architecture.

Manuscript received 15 October 2022; revised 27 December 2022; accepted 3 February 2023. Date of publication 8 February 2023; date of current version 20 March 2023. This work was supported in part by the Italian Ministry of University and Research (MUR) under Project PRIN 2020LWPKH7, in part by the European Union's Horizon Europe Program for Research and Innovation under Grant 101047160, and in part by the Lise Meitner Grants for Israeli–Swedish Research Collaboration under the Project AutoPim. This article was recommended by Guest Editor M. H. Najafi. *(Corresponding author: Esteban Garzón.)*

Esteban Garzón and Marco Lanuzza are with the Department of Computer Engineering, Modeling, Electronics and Systems (DIMES), University of Calabria (UNICAL), 87036 Rende, Italy (e-mail: esteban.garzon@unical.it; m.lanuzza@dimes.unical.it).

Adam Teman and Leonid Yavits are with EnICS Labs, Faculty of Engineering, Bar-Ilan University, Ramat Gan 5290002, Israel (e-mail: adam.teman@biu.ac.il; leonid.yavits@biu.ac.il).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/JETCAS.2023.3243222.

Digital Object Identifier 10.1109/JETCAS.2023.3243222

The contributions of this work are summarized as follows:

- AM^4 is the first NAND-type CAM based on a random-access magnetoresistive crossbar.
- AM^4 is the first solution that converts an MRAM crossbar into a massively parallel, general purpose digital in-memory computer.
- The proposed AM^4 architecture enables multiple applications including CAM, TCAM, approximate CAM, and in-memory associative processor within the same basic design.
- We thoroughly evaluate AM^4 under both software and circuit simulation, and conduct a rigorous design space exploration, including susceptibility to process variations.

The rest of this work is organized as follows: Section II overviews the background of this work; Section III introduces the proposed MRAM crossbar based associative memory/processor design; Section IV presents the functional verification of AM^4; Section V presents the simulation results, while Section VI and Section VII show application results and related work, respectively. Finally, Section VIII summarizes the main conclusions of this work.

II. Background

A. Samsung's MRAM Crossbar Array

Samsung recently unveiled an MRAM crossbar design for in-memory computing [6], which was used to implement a binary neural network. The crossbar array and a schematic representation of a 2T2R MRAM cell are illustrated in Fig. 1(a) and (b). Each crossbar cell comprises two magnetic-tunnel junction (MTJ) devices, which store the data bit and its complement, and two selector transistors (INx and INy), which are driven by signals $\mathrm{IN_j}$ and its complement ($\overline{\mathrm{IN_j}}$) that are shared by all cells in a row j. The bottom node of each cell is connected to the top node of the cell immediately below to enable writing to the crossbar. To complete the write connectivity, an additional switch ($\mathrm{WEN_i}$) connects two vertically adjacent cells to the vertically routed $\mathrm{W_{DATA}}$ signals, as shown in Fig. 1(c). These connections are interleaved, such that even rows are connected to $\mathrm{W_{DATA[0]}}$ and odd rows are connected to $\mathrm{W_{DATA[1]}}$. To write a value into the 2T2R bitcell, the WEN switches that are adjacent to the target cell (above and below) are enabled, and a two-cycle operation is applied. During the first cycle, the INx switch is turned on and the $\mathrm{W_{DATA}}$ signals are driven to write the parallel or anti-parallel states into the left MTJ, denoted as MTJ1 in Fig. 1(c). The same procedure is applied during the second cycle, in which the INy switch is enabled to write the complementary state into the right MTJ (MTJ2).

B. Associative Processor

An associative processor (AP) is a non-von Neumann computer [13]. A 9T static CMOS CAM-based AP is illustrated in Fig. 2. The main component of an AP is an associative memory array (CAM), which allows (1) comparing the entire dataset to a search data pattern (refer to COMPARE KEY in Fig. 2), (2) tagging the matching rows (TAG circuitry is shown in Fig. 2(b)), and (3) writing another data pattern (WRITE KEY in Fig. 2) to all tagged rows. An AP performs no computations in a conventional sense, as no dedicated arithmetic logic units (ALUs) are provided [14]. Instead, arithmetic operations are broken down into a series of Boolean logic equations, which are evaluated by the AP in-memory, in a perfect induction-like fashion, as follows. The dataset is stored in the associative memory, in the input field, typically one data element per CAM row (comprising a virtual Processing Unit, PU, as shown in Fig. 2(a)). The AP matches all possible input combinations of a Boolean function against the input field (for the entire dataset in parallel). During each iteration, the CAM rows containing the matching data elements are tagged, and the corresponding function values (precalculated and embedded in the AP microcode), are written into the designated output fields of the tagged rows. During compare and parallel write cycles, the input and output fields are selected by the MASK register of Fig. 2.

For an m-bit argument x ($x \in$ dataset), any Boolean function $b(x)$ has at most 2^m input combinations. Therefore, a perfect induction-like evaluation of any Boolean function with an m-bit argument would incur up to $O(2^m)$ cycles on an AP, regardless of the dataset size. This can lead to significant performance gains when applied to large datasets.

The main associative instructions (primitives) are:

1) **Compare** ($y_1 == x_1$, $y_2 == x_2, \dots, y_m == x_m$)**:** Compares the query (key) x_i ($1 \le i \le m$) to the field y_i in all rows of the AP array, in parallel. The rows where x_i equals y_i are tagged;
2) **Write** ($y_1 \leftarrow x_1$, $y_2 \leftarrow x_2, \dots, y_k \leftarrow x_k$)**:** Writes the value x_i ($1 \le i \le k$) into the position y_i in all tagged AP rows in parallel.

Arithmetic operations can be performed on an AP in a word-parallel, bit-serial manner, reducing compute time from $O(2^m)$ to $O(m)$. For instance, vector addition may be performed as shown in Fig. 3: Let AP bit-columns 0-3 and 4-7 hold four-bit vectors **A** and **B**, respectively. Columns 8-11 are reserved for the lower 4-bits of the sum vector **S**, while bit column 12 is used for storing and updating the carry bit c (after the addition is complete, it holds the MSB of the vector **S**). The addition is carried out in four single-bit iterations (refer to Algorithm 1), in parallel for all vector elements in the AP:

Algorithm 1 Example of Vector Addition in an Associative Processor

```
For all rows in parallel:
1: for $i = 0$; $i < 4$; $i$++ do
2:     $s_i = \overline{a_i} \cdot \overline{b_i} \cdot c + \overline{a_i} \cdot b_i \cdot \overline{c} + a_i \cdot \overline{b_i} \cdot \overline{c} + a_i \cdot b_i \cdot c$;
3:     $c = \overline{a_i} \cdot b_i \cdot c + a_i \cdot \overline{b_i} \cdot c + a_i \cdot b_i \cdot \overline{c} + a_i \cdot b_i \cdot c$;
4: end for
   (2) and (3) are performed in parallel
   $i$ is the bit index and $c$ and $s$ are, respectively, the carry and sum bits;
```

A single-bit addition iteration is carried out in eight steps, where in each step, one entry of the truth table (a three-bit input pattern, A, B, C_{IN} in Fig. 3(a)) is compared against the

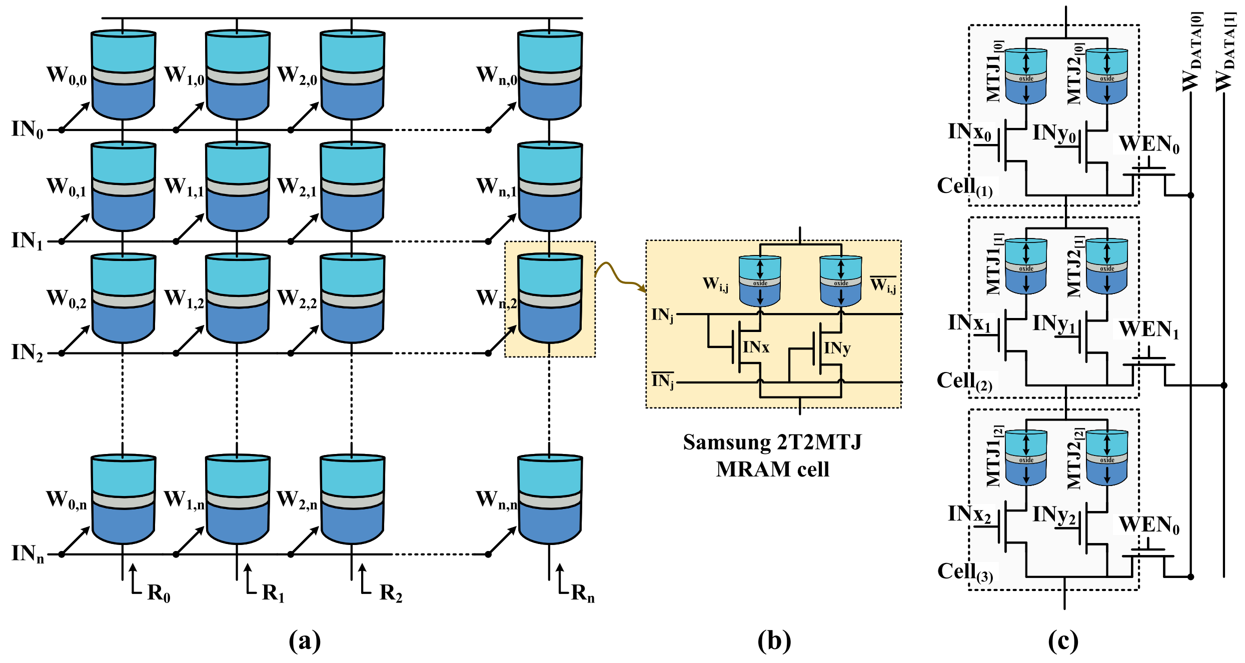


Fig. 1. (a) MRAM Crossbar array. (b) Samsung's 2T2R MRAM cell. (c) Additional data lines for writing into the array. Figures redrawn from Samsung's work [6].

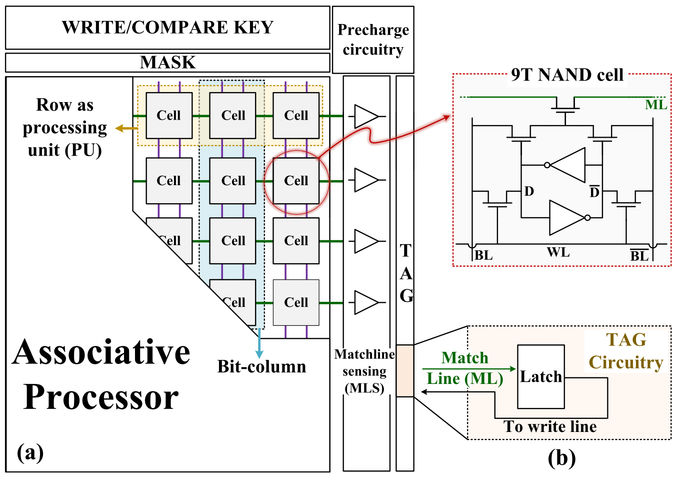
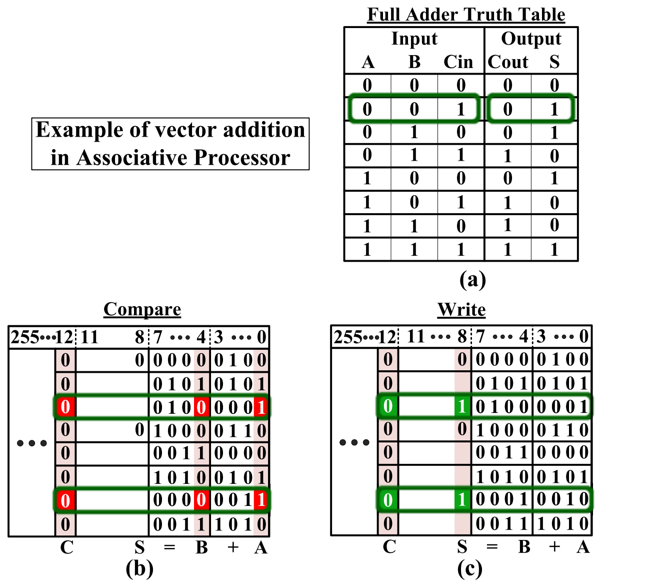


Fig. 2. (a) In-memory associative processor architecture. At the top-right, the 9T NAND CAM cell comprising a 6T-SRAM cell and a wired NMOS XNOR. (b) TAG circuitry. BL is a Bit Line (used also as a Select Line during compare); ML is a Match Line.



Example of vector addition in Associative Processor
Full Adder Truth Table
Compare
Write
(a) (b) (c)

Input			Output	
A	B	Cin	Cout	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Compare

255···12	11 8	7 ··· 4	3 ··· 0
0	0	0 0 0 0	0 1 0 0
0		0 1 0 1	0 1 0 1
0		0 1 0 0	0 0 0 1
0	0	1 0 0 0	0 1 1 0
0		0 0 1 1	0 0 0 0
0		1 0 1 0	0 1 0 1
0		0 0 0 0	0 0 1 1
0		0 0 1 1	1 0 1 0
C	S =	B +	A

Write

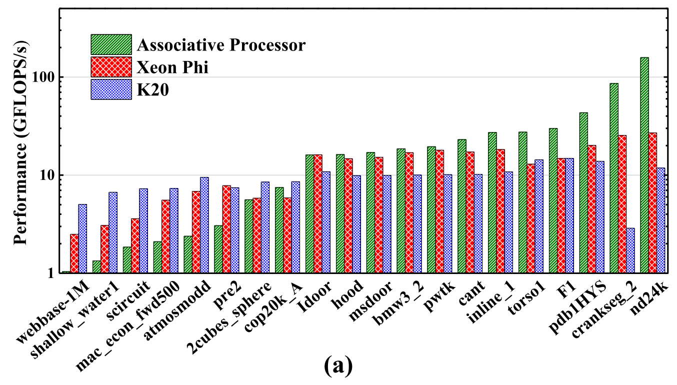
255···12	11 ··· 8	7 ··· 4	3 ··· 0
0	0	0 0 0 0	0 1 0 0
0		0 1 0 1	0 1 0 1
0	1	0 1 0 0	0 0 0 1
0	0	1 0 0 0	0 1 1 0
0		0 0 1 1	0 0 0 0
0		1 0 1 0	0 1 0 1
0	1	0 0 0 1	0 0 1 0
0		0 0 1 1	1 0 1 0
C	S =	B +	A

Fig. 3. Example of vector addition in an associative processor, for two 4-bit vectors **A** and **B**, snapshot at zero bit, second entry of the truth table: (a) Full Adder Truth Table, (b) Compare, only c, a_0 and b_0 are affected, (c) Write, only c and s_0 in the tagged rows (PUs) are affected.

contents of the $a_i \cdot b_i \cdot c$ bit columns and the matching rows are tagged; the logic result (two-bit output (C_{OUT}, S) of the truth table as listed in Fig. 3(a)) is written into the s_i and c bits of all tagged rows.

A snapshot of the second step (processing the second entry of the truth table) of the first iteration (processing LSBs of all vector elements in parallel) is presented in Fig. 3(b)-(c). Fig. 3(a) shows the truth table with the second entry delineated. Fig. 3(b) and (c) show compare and write operations, respectively. During compare (Fig. 3(b)), the input pattern '001' is compared against bit columns a_0, b_0 and c for all vector elements in parallel. The matching rows (two in this example) are tagged. During write (Fig. 3(c)), the output pattern '01' is written into bit columns s_0 and c, respectively. Only the tagged rows are written. Each compare and write affects the entire dataset (vectors **A**, **B** and **S**).

In a straightforward implementation of a Boolean function evaluation, every compare is typically followed by a write operation. This could adversely affect the overall performance and energy-efficiency, since a write may be more time- and energy-consuming than a compare operation. Additionally, this reduces the memory lifetime of write endurance-limited memories. We mitigate the parallel write overhead by using the



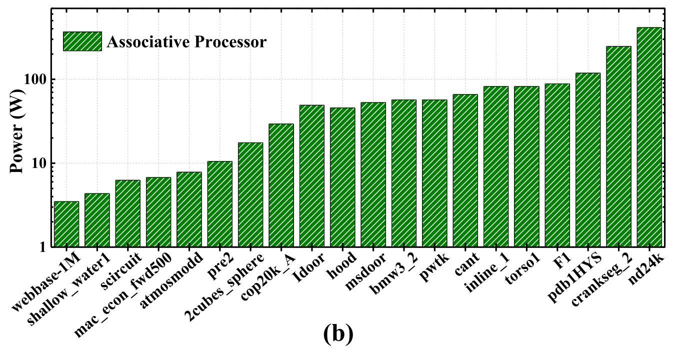


Fig. 4. Sparse Matrix Multiplication [5], [17]: (a) performance, and (b) power.

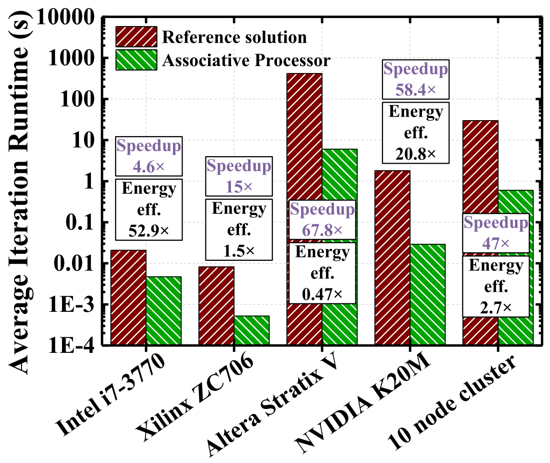


Fig. 5. k-means runtime (lower is better) and energy-efficiency results [18] versus different reference solutions: Intel i7-3770, Xilinx ZC706, Altera Stratix V, NVIDIA K20M, and a 10 node GPU cluster.

observation that any Boolean function has only two values (‘0’ and ‘1’). Therefore, regardless of the truth table size, we can aggregate all compares that are followed by write ‘0’ and write ‘1’ into separate groups, and thus perform a single write per multiple compares [15].

To summarize the complexity of performing typical ALU operations in an AP, a fixed-point addition and subtraction takes $O(m)$ cycles, whereas fixed-point multiplication and division require $O(m^2)$ cycles, where m is the wordlength. Applying a single-precision floating point multiplication on an entire dataset takes 4,400 cycles, regardless of the dataset size [16].

The scaling of CMOS associative processors [16], [17] is limited due to the CMOS CAM density. However, an MRAM AP cell is at least an order-of-magnitude smaller, thus paving the way for associative in-memory computing at scale.

C. Application Space

An AP is a general-purpose computer, whose efficiency strongly depends on the workloads and datasets. As typical for a single instruction, multiple data (SIMD) architecture, the efficiency of an AP is limited in control-flow workloads, but grows quickly for regular iterative workloads with fine-grained parallelism. Since in many algorithms, AP execution time does not depend on the dataset size (as in the above example of vector addition), the efficiency of an AP typically improves with the dataset sizes. Since an AP is an in-memory computer, it is more efficient when running data-intensive applications. Due to the fact that an AP typically implements bit-serial (but word-parallel) arithmetic, it intrinsically supports flexible and user-configurable data wordlengths and formats, including fixed and floating point with flexible mantissa and exponent sizes.

Data in associative memory is accessed by its contents rather than its address. Data elements of the same dataset are normally identified by a unique index, or a member ID. Unlike random access memories (SRAM or DRAM), APs do not require dense and structured data allocation to operate efficiently. Individual data elements do not have to be placed in any specific order but can rather be scattered across random locations (rows) within the CAM arrays. Since modern datasets become increasingly sparse, the ability of computers to properly process sparse data (for example, not wasting time and energy on fetching and multiplying zero-data elements) becomes a critical requirement. An AP holds a significant intrinsic advantage in sparse data processing: sparse data in one of the compressed formats (such as compressed sparse row or compressed sparse column) can be processed almost as efficiently as dense data [17].

A variety of applications have been suggested for APs, including sparse and dense matrix multiplication [17], graph processing [5], deep learning [14], financial and scientific [13], genomics [15] and others [5]. APs exhibit a performance improvement of up to 300 ×, and an energy-efficiency gain of up to 150 ×, compared to CPU, GPU, FPGA, and ASIC implementations. Some selected results are presented in Fig. 4 for different sparse square matrices [5]. Fig. 4(a) and Fig. 5(b) present the performance and power results, respectively, as compared to two reference baselines, CPU and GPU. Fig. 5 presents the k-means runtime (lower is better) and energy-efficiency results versus a variety of CPU [19], FPGA [20], [21] and GPU [22], [23] implementations. The associative processor solution outperforms state-of-the-art alternatives in terms of both performance and energy-efficiency (in all but one case). Another application is shown in Fig. 6, comparing the Convolutional Neural Network AP implementation with state-of-the-art solutions (CPU and a dedicated in-memory accelerator NeuralCache [24]). Again, the benefit of using an AP for such an application is dramatic.

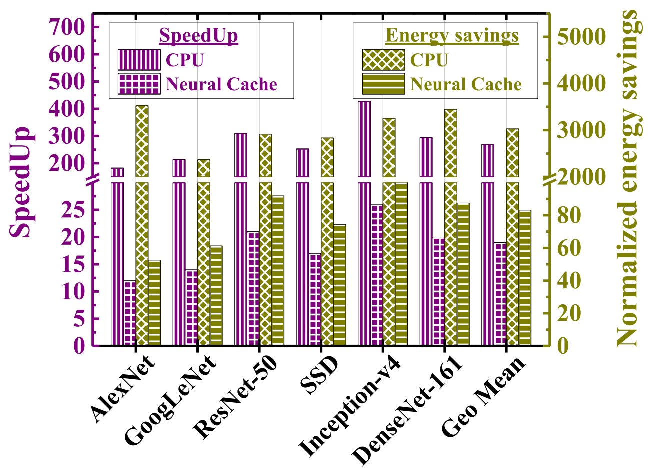


Fig. 6. Speedup and normalized energy savings for a Convolutional Neural Network implementation on an AP [14].

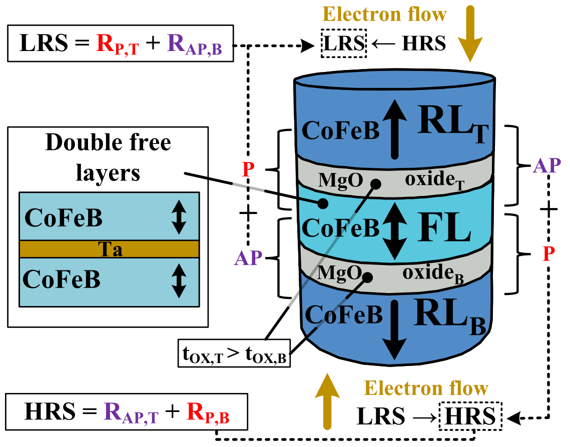


Fig. 7. Structure of a DMTJ featuring two reference layers and two free layers.

TABLE I
Main Physical and Electrical Parameters of the DMTJ With Two Reference Layers and Double Free Layer

Parameter	Description	Value
d	DMTJ diameter	20 nm
t_{FL1}	Thickness FL 1	1.6 nm [25]
t_{FL2}	Thickness FL 2	1 nm [25]
$t_{OX,T}/t_{OX,B}$	Top/bottom oxide thickness	0.85 nm/0.4 nm [25]
I_{c0}	Critical switching current	3.3 µA
HRS	High resistance state	71 kΩ
LRS	Low resistance state	23 kΩ
Δ	Thermal stability factor	57
TMR	Tunnel magnetoresistance ratio at 0 V	≈ 200%

III. MRAM Crossbar Based Associative Memory and Associative Processor Design

A. Double-Barrier Magnetic Tunnel Junction (DMTJ)

Perpendicular MTJs have been widely adopted since 2016, paving their way into stand-alone and embedded memory designs. While standard single-barrier MTJ is the most mature option, it suffers from high writing currents. Using a double-barrier MTJ (DMTJ) with two reference layers is an appealing alternative [11].

The structure of a DMTJ device is shown in Fig. 7. The free layer (FL) is sandwiched between two tunnel oxide barriers ($t_{OX,T}$ and $t_{OX,B}$) with two polarizing reference layers (RL_T and RL_B). Depending on the relative magnetization orientations between the FL and the RLs, the DMTJ features two stable states: the parallel or low-resistance state (LRS), and antiparallel or high-resistance state (HRS). Thanks to the inherently stochastic nature of the spin-transfer torque (STT) switching, the DMTJ can switch between the stable states when a current pulse that is above the critical switching current (I_{c0}) is sent between the top and bottom terminals of the device.

In order to ensure low-energy operation, the AM^4 cell design, introduced in the next subsection, utilizes DMTJ devices as storage elements. We consider an advanced (20 nm diameter) DMTJ structure, whose main bottleneck is the reduced thermal stability factor (Δ), i.e., shorter data retention time [25]. To mitigate this, we target a double FL structure [26]. Therefore, by using a DMTJ with two reference layers and double free layer, we ensure low-energy operation, while maintaining sufficient data retention times.

To simulate DMTJ devices with double free layers, we extended the DMTJ Verilog-A compact model reported in [11]. The model was calibrated with the experimental data reported in [25], [27] by considering the DMTJ parameters shown in Table I.

B. AM^4 Cell

The fundamental idea of AM^4 is transforming the Samsung MRAM crossbar into an associative memory, which can then be operated as CAM, TCAM, ACAM or in-memory AP. The proposed AM^4, therefore, is based on a silicon-proven technology, where the primary novelty is the way that it is operated and utilized. This approach is inspired by CMOS-based CAM built upon a modified 6T CMOS SRAM [28].

The concept of AM^4 is to virtually rotate the 2T2R MRAM bitcell by 90° and repurpose the control signals, as illustrated in Fig. 8. The IN_j and $\overline{IN_j}$ signals are used as Search Lines (SLleft and SLright), where the compare (query) pattern is asserted. The resistive path through the serially connected MRAM bitcells serves as a NAND-style Match Line (ML), enabling the NAND CAM compare functionality. We further label the stored states as '1' and '0' according to the resistances of the left (now top) and right (now bottom) DMTJs. A '0' is stored by writing parallel spin (LRS) into the top DMTJ and anti-parallel spin (HRS) into the bottom DMTJ. The complementary state (HRS, LRS) is considered a '1'.

C. Working Principle

The first of two primary operations of AM^4 is the compare operation. Its implementation in the 2T2R MRAM crossbar-based AM^4 is presented in Fig. 9. Compare is achieved by first precharging the ML and then driving the search pattern onto the SLs (SLright = $\overline{\text{SLleft}}$), connecting the leftmost bitcell in a row to ground, and disabling both the WEN_i and W_{DATA} signals (refer to Fig. 1(c)). If SLleft = 1 (SLright= 0) and

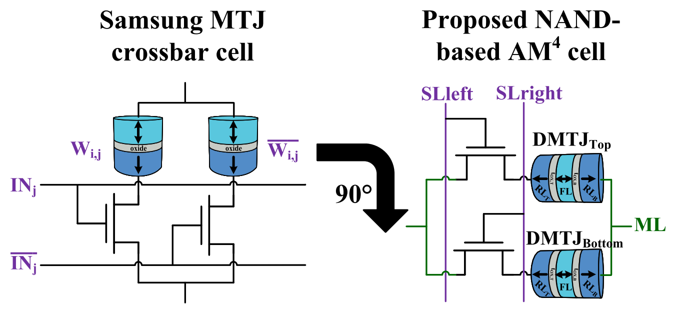


Fig. 8. Rotating and relabeling the 2T2R MRAM cell to transform it into an AM^4 cell. Note that this conceptual rotation and signal repurposing does not require any circuit design or process changes to the silicon-proven crossbar, fabricated by Samsung [6].

the cell is in the ‘0’ state (LRS, HRS), or if SLleft = 0 (SLright= 1) and the cell is in the ‘1’ state (HRS, LRS), the output resistance of the cell is low, representing a match between the query bit of that column and the bit stored in the bitcell. If all cells in a row match, a low resistance is displayed by the row, enabling a fast discharge of the ML to ground. A mismatch occurs when at least one bit of the query pattern does not match the bit stored in the corresponding AM^4 cell. In such a case, the conductance path through the row goes through HRS DMTJ(s), preventing or slowing down the ML discharge. By differentiating between these two options, a per-row compare operation can be achieved.

The other main primitive of AM^4 is a parallel write operation. It is achieved using the $W_{DATA[0]}$ and $W_{DATA[1]}$ signals, which we label TAG and $\overline{\text{TAG}}$, respectively, since they are driven by the AM^4 tags (refer to Fig. 2(b)). As presented previously, writing into the MRAM cell requires two cycles to separately bias each DMTJ for applying the required magnetization orientation. However, due to the sharing of the WEN switches between adjacent columns (which creates a potential sneak path along the bitcell row), a parallel write operation requires four single-cycle phases, as illustrated in Fig. 10 (phases 1 to 4). During the entire write operation, all WEN switches are turned on, and the selection of individual MTJs is done using the SLleft and SLright signals. In phases 1 and 2, the even columns are written. In phases 3 and 4, the odd columns are written. Multiple rows can be written at the same time.

During phases 1 and 3, ‘1’ is written to the top DMTJ for bitcells where ‘1’ is supposed to be stored and to the bottom DMTJ for bitcells where ‘0’ is supposed to be stored. During phases 2 and 4, ‘0’ is written to the top DMTJ for bitcells where ‘0’ is supposed to be stored and to the bottom DMTJ for bitcells where ‘1’ is supposed to be stored.

During phases 1 and 2, SLright is a complement of SLleft (SLright = $\overline{\text{SLleft}}$) in all even columns. The SLleft and SLright signals of the odd columns are both driven to 0, thereby isolating these columns and blocking a potential sneak path through the bitcell rows.

In phase 1, TAG and $\overline{\text{TAG}}$ lines are driven to V_{write} and GND, respectively. The SLleft bit of the even columns, where ‘1’ is supposed to be stored, is asserted to ‘1’ to enable writing ‘1’ to the top DMTJ. The SLleft bit of the even columns, where ‘0’ is supposed to be stored, is asserted to ‘0’ (i.e., SLright is asserted ’1’) to enable writing ‘1’ to the bottom DMTJ. In phase 2, the TAG and $\overline{\text{TAG}}$ lines are swapped; The SLleft bit of the even columns, where ‘0’ is supposed to be stored, is asserted to ‘1’ to enable writing ‘0’ to the top DMTJ. The SLleft bit of the even columns, where ‘1’ is supposed to be stored, is asserted to ‘0’ (i.e., SLright is asserted ’1’) to enable writing ‘0’ to the bottom DMTJ.

After finishing the write operation to the even columns, the same procedure is followed in phases 3 and 4 to write to the odd columns (refer to Fig. 10).

Using the compare and parallel write operations, described above, the Samsung MRAM crossbar-inspired topology can be used “as is”, i.e., with no alterations, to enable CAM, TCAM, ACAM and AP functionalities.

D. AM^4-Based Associative Processor

The AM^4-based associative processor is presented in Fig. 11. Its core is the 2T2R AM^4 crossbar. The columns are supplemented with write/compare key and mask registers above the array. Masking-off (i.e., rendering certain bit columns unaffected by either compare or write), is achieved by setting SLleft = SLright = ‘1’ for compare and SLleft = SLright = ‘0’ for write, respectively. The Matchline Sensing (MLS) column is built with row-connected sense amplifiers, which drive the TAGs, as follows:

1) Sense Amplifier: The matchline sensing (MLS) topology is based on the single-ended self-reference sense amplifer proposed in [29]. The MLS comprises two transistors and two inverters as shown in Fig. 12(a). The precharge transistor (MPC) enables the ML precharge ($\overline{\text{PC}} = 0$) and ML evaluation stages ($\overline{\text{PC}} = 1$). The MX transistor, along with inverter I1, serve to limit the ML precharge to the voltage level of the tipping point of I1. Inverter I2 evaluates the final response of the ML.

Different from the standard operation of the MLS [29], the SL patterns are driven into the AM^4 cell during the evaluation stage. The sensing stage starts by driving the PC signal to GND ($\overline{\text{PC}} = 0$), precharging the ML. The V_X line is charged up to V_{DD}, and ML_{out} is discharged to GND. Similarly, the ML is charged up to a voltage level that depends on the I1 threshold, whose value, when exceeded, drives the gate of MX. This turns off MX, thereby halting the ML precharge. Subsequently, the SL signals are driven to the AM^4 cells, and $\overline{\text{PC}}$ is asserted (‘1’) to start the evaluation stage. The ML starts to discharge quickly or slowly depending on the output resistance of the AM^4 cells. A match is detected when all cells are in LRS (ML_{LRS}), presenting the lowest resistance path in the NAND-based AP word, and therefore, raising ML_{out} to V_{DD}. Conversely, a mismatch occurs when an HRS state is present, slowing down the ML discharge. During the mismatch sensing (ML_{HRS}), the evaluation time-frame is short enough to avoid compare errors, maintaining the ML_{out} signal close to ‘0’. This is shown in Fig. 12(b).

The worst case ML sensing scenario differentiates between a match (all cells are in LRS) and a single-bit mismatch (all cells but one are in LRS). There is an overlap between

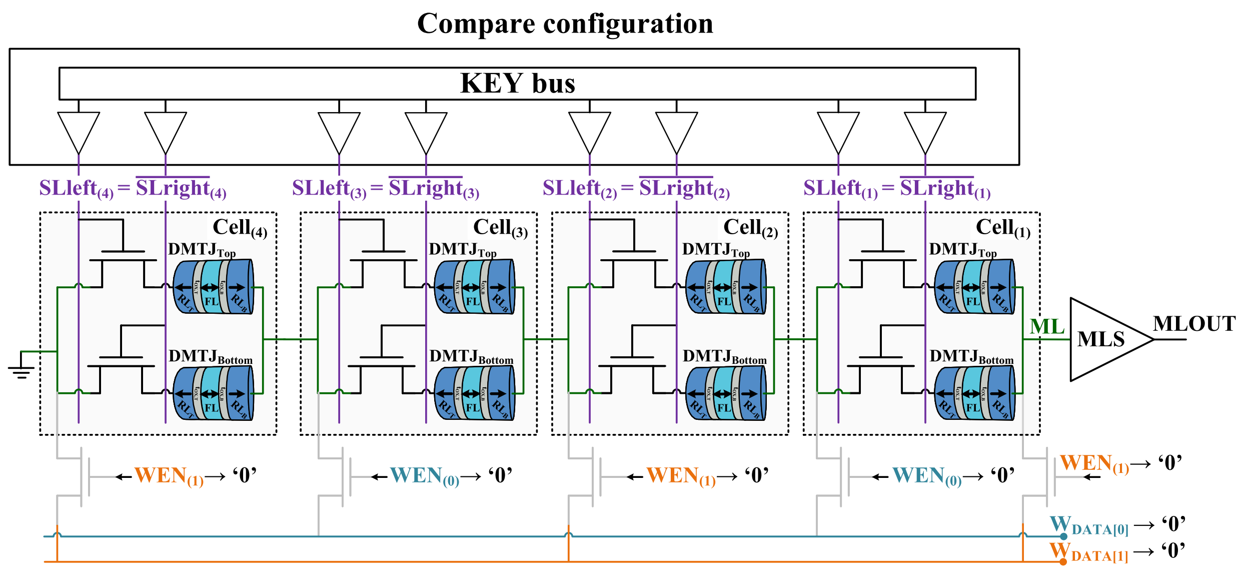


Fig. 9. Compare operation in AM^4. The row connectivity serves as a NAND-style Match Line (ML). The ML is terminated by GND on the left. The ML is connected to a precharge circuit and the ML sense amplifier (MLS) on the right.

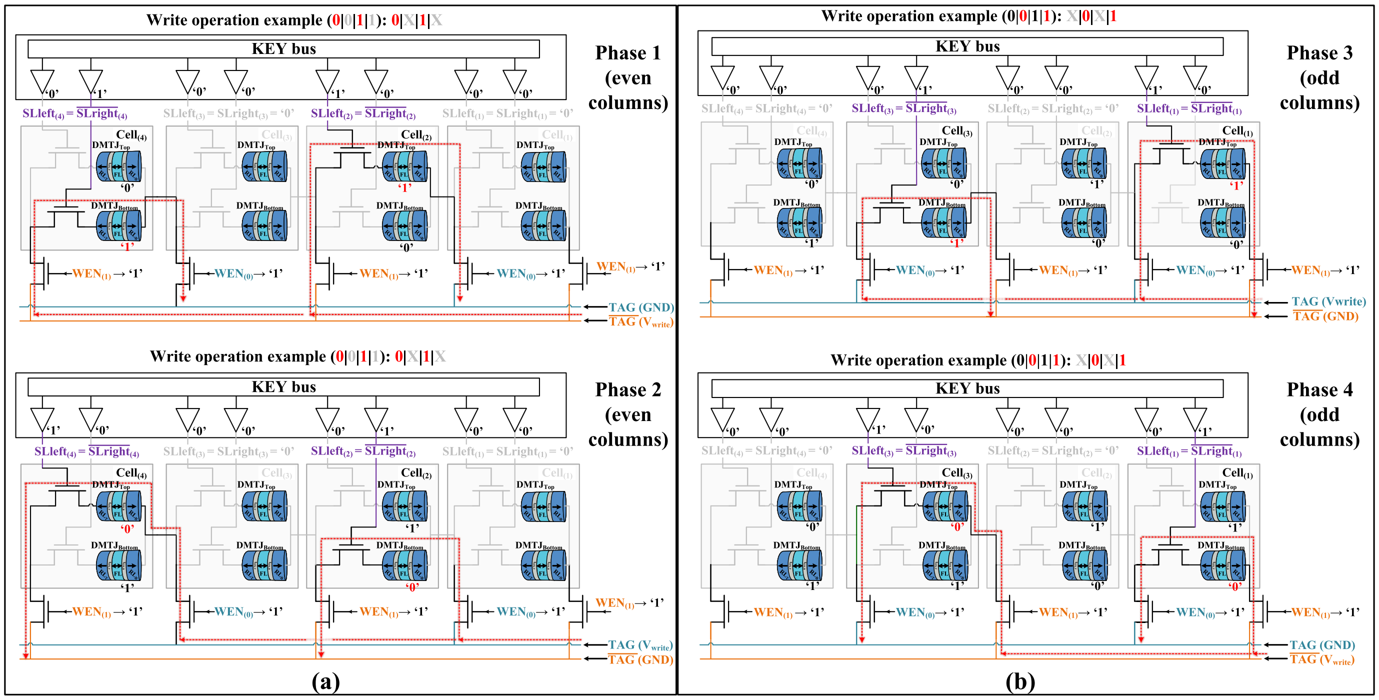

Write operation example (0|0|1|1): 0|X|1|X
Phase 1 (even columns)
Write operation example (0|0|1|1): 0|X|1|X
Phase 2 (even columns)
Write operation example (0|0|1|1): X|0|X|1
Phase 3 (odd columns)
Write operation example (0|0|1|1): X|0|X|1
Phase 4 (odd columns)
KEY bus
TAG (GND)
TAG (Vwrite)
(a)
(b)

Fig. 10. Write operation in AM^4. The write pattern is '0011' (the MSB is written to the leftmost column 4). (a) In phase 1, we write '1' in the top DMTJ of the 2nd column and the bottom DMTJ of the 4th column. In phase 2, we write '0' in the bottom DMTJ of the 2nd column and the top DMTJ of the 4th column. (b) In phase 3, we write '1' in the top DMTJ of the 1st column and the bottom DMTJ of the 3rd column. In phase 4, we write '0' in the bottom DMTJ of the 1st column and the top DMTJ of the 3rd column.

the match and a single-bit mismatch, mainly due to poor HRS/LRS ratio of MRAM. We mitigate this through redundant data coding, specifically by ensuring a certain minimum Hamming Distance (HD) between any two data words. Since contemporary memories typically use Error Correction Codes (ECC) [30], there is a "built-in" minimum Hamming distance which we utilize for the purpose of match vs. single-bit mismatch differentiation. Based on this insight, ECC-protected AM^4 provides a safe margin between match and mismatch cases for a limited HRS/LRS ratio memory. This method is more relevant for exact searching. On the contrary, using ECC might be inapplicable in approximate search scenarios, which fortunately are likely to be less sensitive to inexact matching results.

2) TAG Circuitry: The bottom part of Fig. 11 details the circuitry of the TAG register cells. Its main component is a flip-flop (FF) that holds the compare result according to the control logic signals, i.e., compare and write.

To compare the query (key) data word against the data stored in the AM^4 array (the entire row, a number of bits or a

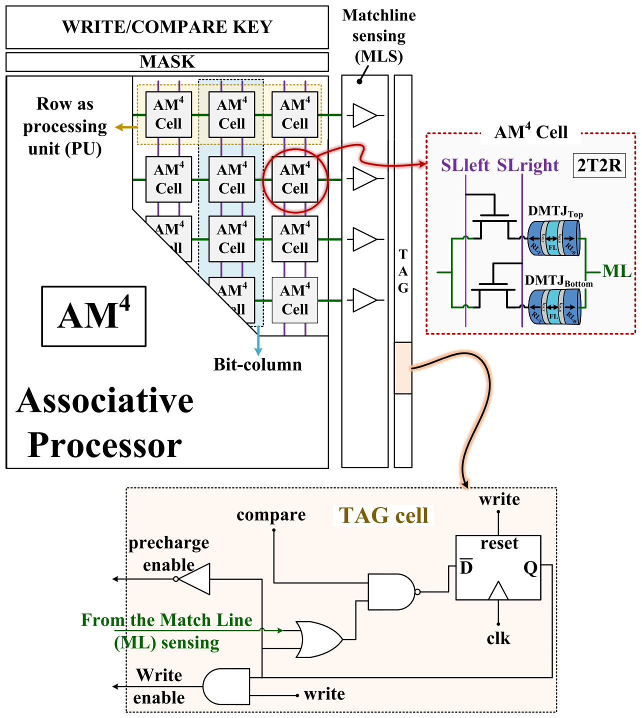


Fig. 11. AM^4 based in-memory associative processor. The ML precharge circuitry is within the ML sensing scheme (MLS).

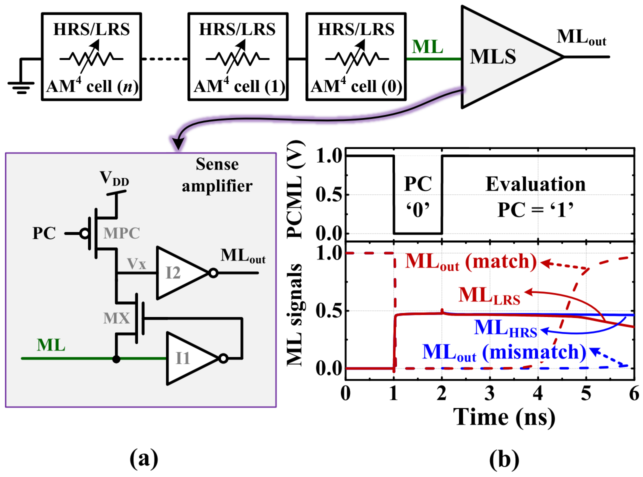


Fig. 12. (a) Matchline sensing structure of AM^4 n-bit row. (b) Timing diagram of the sensing scheme for a 32-bit AM^4 word.

single bit), the ML is precharged, and the key is driven onto the SLs. In order to mask a column (i.e., ignore it during compare), the SLleft and SLright lines are set to '1'. If every unmasked bit in a row matches the corresponding query bit, the ML is discharged and a '1' is accumulated in the TAG FF. If even a single unmasked bit mismatches the corresponding query bit, the ML remains high and the TAG FF remains unchanged.

As detailed in Section II-B, a compare (or several compare) operation(s) in an AP are typically followed by a parallel write into the unmasked bits of all tagged AP rows. To write data from the write key register into AM^4, each TAG FF (set earlier by compare operation(s)) is connected to its corresponding WEN. We accumulate the compare results in each AP row to reduce the number of writes, which improves the performance of the AP. If the result of aggregated compare operations is '1', the write key data set on SL lines is written into the AP row in accordance with the MASK pattern. Otherwise, the write does not affect the row.

E. Content Addressable Memory (CAM) and Ternary CAM (TCAM)

In addition to its functionality as an AP, AM^4 can be operated in CAM/TCAM mode. Operation in CAM mode is straightforward, according to the search and write operations, described previously. To support TCAM mode, either the search (query) pattern bits or the bits of the data patterns stored in the MRAM crossbar can be "don't care" in addition to conventional '1' and '0' values. To store a "don't care", a '0' is written to both top and bottom MTJs of an AM^4 cell, presenting a LRS through both top and bottom discharge paths to the ML. During a compare, the "don't care"-written cell will not affect the result of the operation (match or mismatch). To create a "don't care" search pattern bit, we simply mask it off, as presented above.

F. Approximate Search CAM (ACAM)

Multiple applications, including text processing (e.g., text retrieval, signal processing, computational biology [31], [32], [33], and genome analysis [9], [15]), require approximate rather than exact search, for example to tolerate errors, or find similarities among erroneous or ambiguous data patterns. In approximate search, if the difference between a stored pattern and the query pattern is below a certain predefined threshold, the compare result should still be considered a "match". AM^4 can support approximate search by adjusting the MLS sampling time, using the speed of the matchline discharge as a measure of Hamming distance.

To make the operation mode robust, an ML can be amended by an NMOS discharge transistor with a configurable gate voltage. In such a configuration, the approximate search utilizes the matchline charge redistribution rather than its rise or fall time. By tuning this gate voltage (possibly automatically), we can set a desired level of Hamming distance without adjusting the ML sampling time [34], [35].

IV. Compare and Write Functional Verification

To validate the write and compare operations we considered a 32×32 array, operating with a supply voltage (V_{DD}) of 1 V at nominal conditions.

A. Write Operation

Fig. 13 shows the simulation waveforms of write '0', '1', and 'X' operations. As an initial condition, we set all DMTJ devices to HRS. The write key (pattern) is $(000XXFFF)_{Hex}$, i.e., top and bottom MTJs are written to be $(00000FFF)_{Hex}$ and $(FFF00000)_{Hex}$, respectively.

WEN signals are enabled during four phases of the write operation (refer to Section III-C). An extra phase is added at the end of phases 2 and 4 to write the "don't care" value (as required in TCAM). The write key bus drives the write

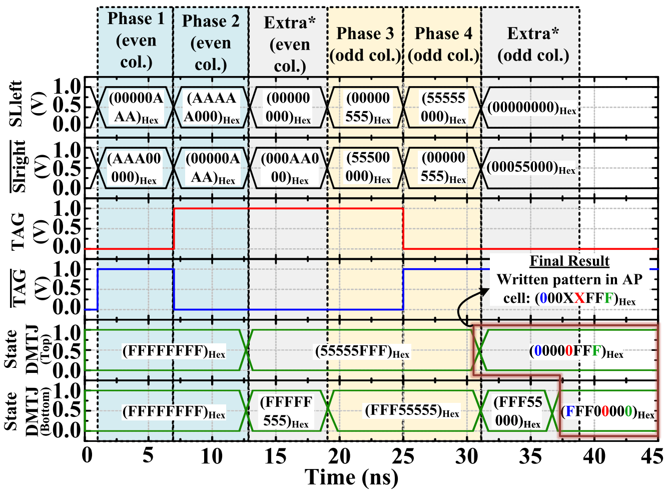


Fig. 13. Functional verification of AM^4 write operation. The write operation refers to a 32-bit row.

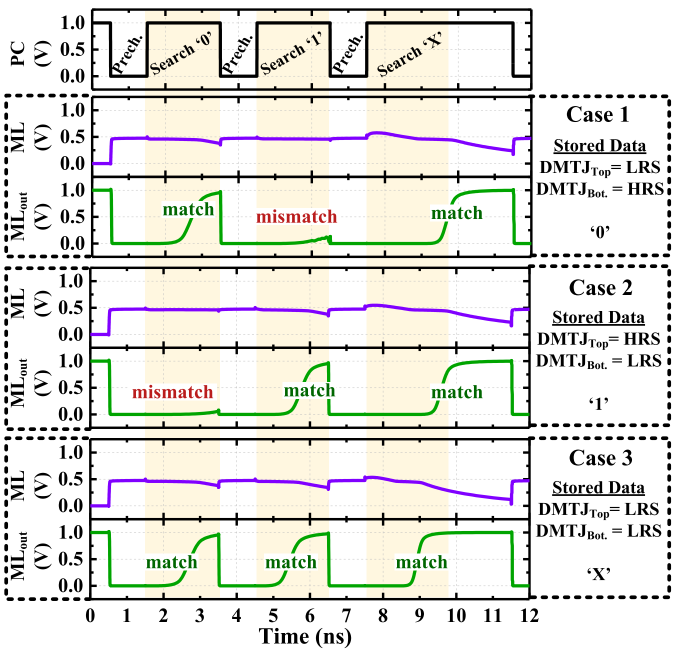


Fig. 14. Functional verification of AM^4 compare operation in a 32-bit AM^4 row. Search '0', '1', and 'X' operations are evaluated with respect to all possible stored values (Case 1–'0', Case 2–'1', and Case 3–'X'.)

pattern to SLleft and SLright. As a result, top and bottom DMTJs are set into the desired stable resistance state. The final result is highlighted at the end of the write operation. For the sake of clarity, the top and bottom DMTJs states are also highlighted. Refer to the colored patterns within the 37 ns-45 ns time-frame. The AM^4 cell in '0', 'X', and '1' states correspond to DMTJ$_{\text{Top}}$ (DMTJ$_{\text{Bottom}}$) in '0' ('1'), '0' ('0'), and '1' ('0'), respectively. Therefore, the write operation is verified.

B. Compare Operation

Fig. 14 shows the AM^4 compare operation involving all possible stored values, i.e. '0', '1', and 'X'. We consider a compare time of about 2 ns (see Section V). In the precharge stage, the ML is precharged to about half V_{DD} (see Section III-D.1), and ML$_{\text{out}}$ is discharged to '0'. Then, during the evaluation stage, SL signals are assigned. When comparing with '0', '1', and masking out the entire AM^4 row (which is equivalent to comparing with an 'X'), the SLleft/SLright pattern is (FFFFFFFF)$_{\text{Hex}}$/(00000000)$_{\text{Hex}}$, (00000000)$_{\text{Hex}}$/(FFFFFFFF)$_{\text{Hex}}$, and (FFFFFFFF)$_{\text{Hex}}$/(FFFFFFFF)$_{\text{Hex}}$, respectively. In the match (mismatch) case, the LRS (HRS) path discharges the ML (maintains the ML at approximately half V_{DD}), and the ML$_{\text{out}}$ outputs a '1' ('0'). If a certain AM^4 cell stores an 'X', or a certain bit column is masked out (by setting SLleft = SLright = '1'), such a cell keeps both top and bottom paths at LRS (open) and hence does not affect the outcome of compare.

Overall, the compare operation was successfully demonstrated with all possible stored data, i.e., '0', '1', 'X'.

V. Circuit-Level Results

Results provided in this section rely on extensive Monte Carlo simulations (1000 samples) at the 3σ corner probability distribution from the statistical models given by the 28 nm FDSOI commercial PDK. For the DMTJ devices, the effect of process variability follows Gaussian-distributed variations with a variability (σ/μ) of 5% for the DMTJ cross-section areas, and 1% for $t_{\text{OX,T}}$, $t_{\text{OX,B}}$, and t_{FL} [36].

Fig. 15 shows the compare timing for a 32-bit AM^4 row operating at a supply voltage of 1 V at the Typical-Typical corner. In particular, ML$_{\text{out}}$ responses to match and several mismatch cases are shown. ML$_{\text{out}}$ is simulated over the evaluation time-frame t_{comp} ranging from 1 ns to 3 ns. The wider the evaluation time-frame t_{comp}, the longer the ML has to discharge through the NAND-style resistive path, eventually driving ML$_{\text{out}}$ to '1' (which results in a compare error). For example, if the compare evaluation stage extends to 3 ns, the 4-bit mismatch will register as a "match" rather than a "mismatch" (thus creating a compare error). This happens because of the poor DMTJ HRS/LRS ratio.

As presented above, we mitigate the effects of the limited HRS/LRS ratio by data coding which ensures a certain minimum HD between datawords. A minimum HD, h, guarantees that the lowest number of mismatching bits in the worst case mismatch equals h rather than 1. Since ECC schemes are regularly included in contemporary memories, especially in nonvolatile ones [30], a certain HD is typically maintained. Hence, we do not necessarily have to extend the memory redundancy to introduce minimum HD. To identify the minimum HD required for safe AM^4 operation, the compare times of the match and several mismatch cases are analyzed through Monte Carlo simulations.

Fig. 15(b) shows the statistical distribution of t_{comp} for the highlighted cases of Fig. 15(a), i.e., match and 12-bit mismatch. We define the t_{comp} difference between match and mismatch at nominal and 3σ conditions, as δ_{μ} and $\delta_{3\sigma}$, respectively. δ_{μ} gives a difference of about 870 ps, and at the 3σ corner this difference is reduced by 2.5 ×. Nevertheless,

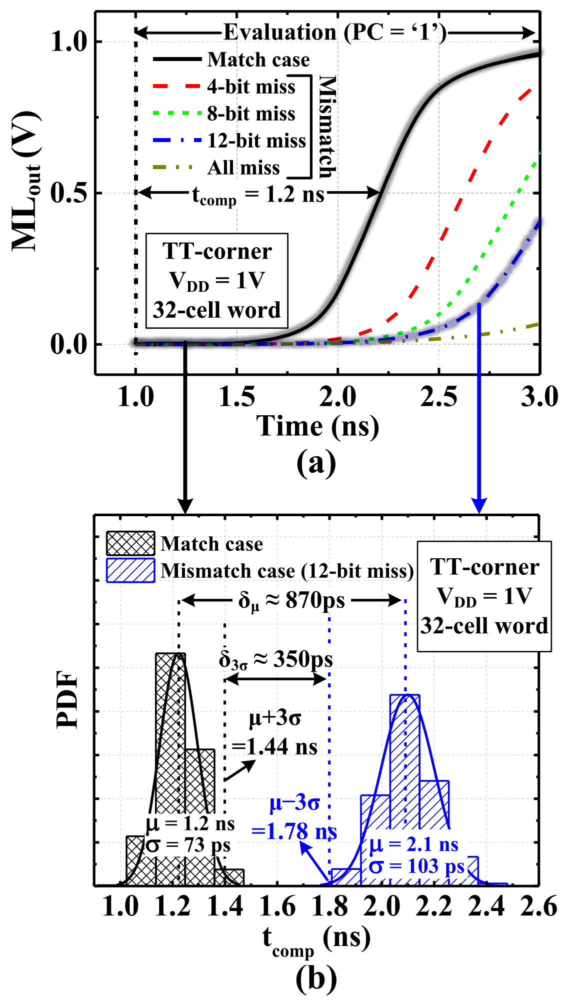


Fig. 15. Compare time results of a 32-bit AM^4 row at the Typical-Typical (TT) corner with V_{DD} of 1 V. (a) Nominal results: match line response for a match and different mismatch cases. (b) Monte Carlo results: Statistical distribution of the compare time of match and 12-bit mismatch cases for 3σ corner evaluation. Results are from 1000 Monte Carlo simulations.

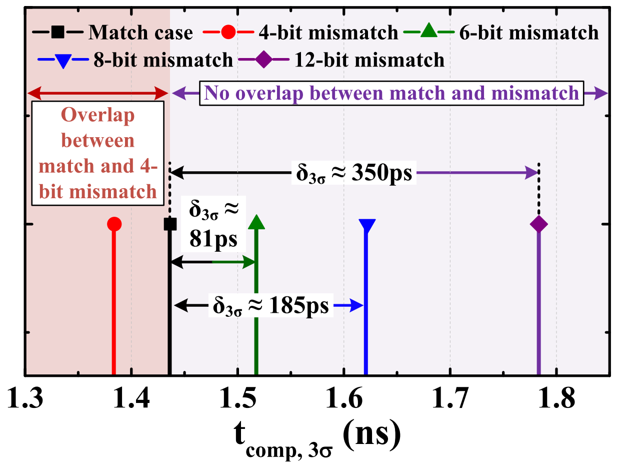


Fig. 16. Compare time results for different match and adjacent mismatch cases. Results are from 1000 Monte Carlo simulations at 3σ and TT-corner with a V_{DD} of 1 V.

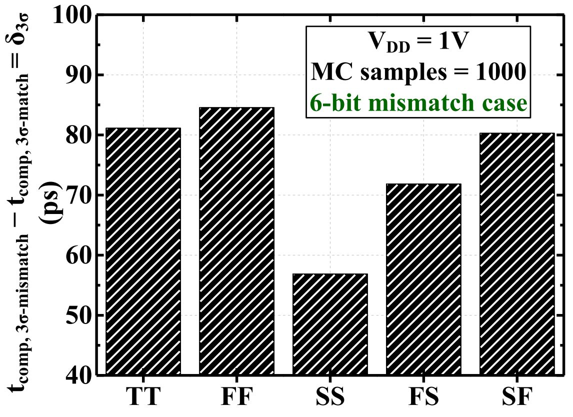


Fig. 17. AM^4 susceptibility to process variations: Compare time for the Typical-Typical (TT), Fast-Fast (FF), Slow-Slow (SS), Fast-Slow (FS), Slow-Fast (SF) corners.

these Monte Carlo results show that 3σ values do not overlap, suggesting that AM^4 performs correctly at a t_{comp} of 1.4 ns. In this example, we show that t_{comp} results remain within a safe time-frame region, assuming the minimum HD between any two data words is 12-bit. To identify the effective minimum HD allowed by a 32-bit AM^4 word, we repeat the same Monte Carlo simulations for different match and mismatch cases.

Fig. 16 shows the compare time analysis through Monte Carlo simulation at the 3σ corner. For a t_{comp} time of 1.44 ns, AM^4 operates correctly during compare operations, properly differentiating between match and a 5-bit and above mismatch. This result is obtained while also ensuring a $\delta_{3\sigma}$ of about 100 ps. As shown in Fig. 16, the overlap region (refer to red time-frame at the left) between the match and 4-bit mismatch suggests that a minimum HD distance of 4 may be always required to avoid compare errors. To ensure the minimum HD of at least 4, we may use one of the error correcting codes, such as BCH. For example, BCH(31,21,2) code guarantees the minimum HD of 2×2.1=5. While a 32-bit wide AM^4 suffices for a number of applications [5], it is reasonable to assume that for certain other applications, a wider AM^4 array will be required. Due to the low HRS/LRS ratio of the MTJs, a wider AM^4 row leads to a higher compare error probability. In such a case, the minimum HD should be increased accordingly.

Lastly, we evaluate the the impact of local variations on t_{comp} at 3σ, based on the match and 6-bit mismatch from the above analysis (refer to Fig. 16). The local variations are around even corners, i.e., Typical-Typical (TT), Fast-Fast (FF), and Slow-Slow (SS), and skewed corners, i.e., Fast-Slow (FS), and Slow-Fast (SF), as shown in Fig. 17. The $\delta_{3\sigma}$ results present robustness to local variations, mainly because of the adopted single-ended sensing scheme. The same behavior was presented for the other mismatch cases.

Table II summarizes the main circuit-level results for a 32-bit wide AM^4 array operating at a nominal voltage of 1 V. The reported data is obtained through Monte Carlo simulations for both write and compare operations, evaluated at the 3-sigma corner. AM^4 compare operation consumes 1.73 fJ per bit, and presents a compare time of 1.44 ns. A write latency of 6.68 ns is mainly due to the DMTJ resistances in HRS and LRS; it can be reduced by increasing the access transistor size. However, since we aggregate compare results

TABLE II
SUMMARY RESULTS FOR CIRCUIT-LEVEL ANALYSIS OF THE NAND-BASED ASSOCIATIVE PROCESSOR

Parameter	Value
AP topology	NAND-based
Technology	28 nm FDSOI
V_{DD} (V)	1
Word size (bits)	32
t_{comp} (ns)	1.44
$E_{compare}$ (fJ/bit/compare)	1.73
t_{write} (ns)	6.68
E_{write} (fJ/DMTJ)	85.8
Area (μm^2)	0.138

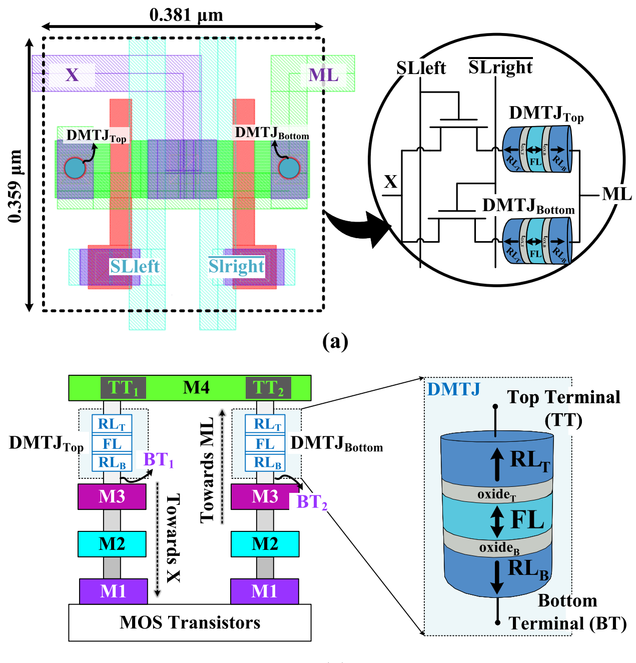


Fig. 18. (a) Layout of the AM^4 cell. (b) 3D hybrid CMOS/DMTJ process. MTJs are placed between Metal-3 (M3) and Metal-4 (M4).

to reduce the number of writes, the effect of write latency on AM^4 performance is minimal.

Table II also reports the area of the AM^4 cell, whose layout and an illustration of the 3D hybrid CMOS/DMTJ process, are shown in Fig. 18(a)-(b). The inclusion of the access transistor for write operation (refer to Fig. 10) would incur an overhead of about 48% of the cell area. This access transistor is also presented in the Samsung crossbar cell [6].

VI. Application of AM^4 AP to Smith-Waterman DNA Sequence Alignment

Smith-Waterman is a dynamic programming algorithm [12] that identifies the optimal alignment of two sequences. It is widely used in bioinformatics and computational biology for DNA (genome) sequence alignment. Smith-Waterman algorithm has two steps. The first step is scoring, which builds a two-dimensional scoring matrix to find the maximum edit distance between two sequences. The second step is a traceback, which reconstructs the optimal alignment path. Scoring is the most computationally demanding step [15], while traceback requires significantly less computing power and can therefore be performed by an external host CPU. In the following, we focus on AM^4-based AP implementation of the scoring step.

The sequential time complexity of the Smith-Waterman score matrix calculation is $O(nm)$ where n and m are the lengths of both sequences. The upper bound of the Smith-Waterman scoring complexity on a parallel von Neumann machine with p parallel processing units is $O(nm?p)$. AM^4 based AP can achieve linear time complexity of $O(\max(n,m))$. Smith-Waterman is used in genome analysis to find the optimal local alignment (of two or more DNA sequences). Global alignment, which is also a frequently used genome analysis tool, can be implemented on an AM^4-based AP with only a few modifications to the local alignment implementation. Similarly, an AP can efficiently implement a multiple sequence alignment [15].

We compare AM^4 with state-of-the-art sequence alignment solutions SWAPHI-LS [37], RIVYERA [38], CUDAlign 4.0 [39], PRINS [9], and SWhybrid [40]. For evaluation, we used six genomes from the publicly available NCBI nucleotide database [41], with lengths varying from 4.4M basepairs[1] (bps) up to 50M bps. Performance of the Smith-Waterman algorithm was measured in Cell Updates per Second (CUPS). As shown in Fig. 19, AM^4 outperformed (was more energy-efficient than) SWAPHI-LS by 41 × (125 ×), RIVYERA by 3.3 × (2.2 ×), CUDAlign 4.0 by 1.6 × (176 ×), and Swhybrid by 4.9 × (11 ×). As for the PRINS reference solution, AM^4 is about 12% slower, mainly due to the multiple number of cycles during a write operation. Nevertheless, AM^4 is still more energy-efficient (1.5 ×) than PRINS.

Our comparison includes the resistive device-based NOR-type AP (RAP) presented in [42]. NOR configuration is the reason RAP outperforms the AM^4. However, for the same reason, RAP achieves significantly lower energy efficiency (–69%) compared to AM^4.

For the above examined workload, AM^4 performance is limited by the density of the datasets and the parallelism of the task. Smith-Waterman DNA sequence alignment scoring matrix calculation is an example of a highly parallelizable task where a large number of data elements are processed simultaneously, allowing AM^4 to apply its parallel processing abilities. In a Smith-Waterman workload, AM^4 scales to the dataset size.

VII. Related Work

A. Content Addressable Memory

Several ternary and binary CAM designs have been proposed in recent years, including CMOS-based [16], [43], [44], [45], [46], [47], [48], [49], [50], [51], [52], as well as emerging memory based [5], [9], [13], [53], [54] solutions. Several emerging memory (memristor crossbar) approximate search CAM designs have also been proposed [55], [56]. Some offer soft-error tolerance using error correction coding (which requires memory redundancy) and replacing the matchline

[1]DNA nucleotides (Adenine (A), Guanine (G), Cytosine (C), and Thymine (T)) are frequently referred to as DNA basepairs, bases or bps.

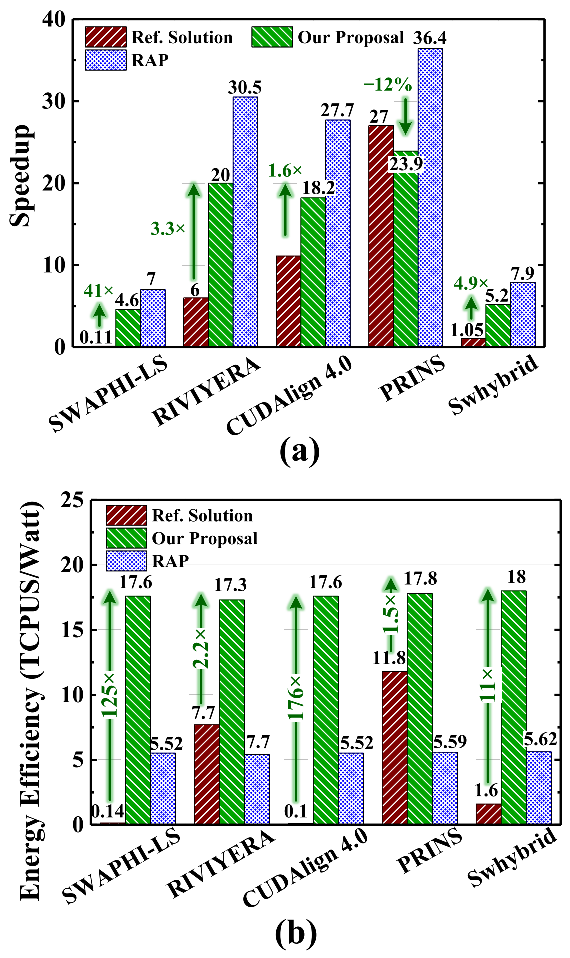


Fig. 19. Smith-Waterman DNA and Protein Sequence Alignment: (a) speedup and (b) energy-efficiency gain of AM^4 and the design (RAP) proposed in [42] versus different reference solutions: SWAPHI-LS [37], RIVYERA [38], CUDAlign 4.0 [39], PRINS [9], and SWhybrid [40]. Higher is better; TCUP = Tera cell updates per second.

sense amplifier with an analog comparator [57], [58]. These designs typically tolerate only a limited Hamming distance (1-4 bits) [59], [60].

A variety of approximate search CAM designs use timing (i.e., score signal delay or the speed of the matchline discharge) as a measure of Hamming distance. Bui and Shibata [61] exploits the delay of the score signal for a Hamming distance search CAM. A small Hamming distance tolerance ($\leq$ 2 bits) approximate CAM is proposed in [62], [63]. Garzón et al. [34] and Hanhan et al. [35] use the combination of the voltage, controlling the speed of the matchline discharge, and the sense amplifier reference voltage to define the Hamming distance threshold. These designs are capable of tolerating very large Hamming distances.

AM^4 differs from conventional and approximate CAM solutions in that to our knowledge, it is the first NAND-type CAM based on a random-access magnetoresistive crossbar. Typical state-of-the-art emerging memory based CAMs are designed as NOR CAM, where the matchline discharges on a mismatch occurrence. Since mismatches are much more frequent than matches (in typical CAM/TCAM applications, only one memory row matches, while the rest mismatch), all matchlines need to be pre-charged before every search/lookup, resulting in significant energy wasting. In contrast, in NAND CAM, only the matching row(s) discharge, reducing the energy consumption of search/lookup by orders of magnitude.

B. Associative Processor

The use of STT-MRAM and Resistive Ternary CAM (TCAM) for data-intensive computing was proposed by Guo et al. [64]. ReAP, a resistive memory based, massively parallel in-memory associative processor was first introduced by Yavits et al. [13]. Yantir et al. [65] introduced a two-dimensional model of an in-memory associative processor. Hout et al. [66] extended the associative processor model to support multi-valued logic. Imani and Rosing [67] proposed another design of an associative processor for near-memory processing. Caminal et al. [68] applied in-memory associative processing to database analytics acceleration, while Yavits et al. [17] and Neggaz et al. [69] implemented in-memory matrix multiplication on an associative processor. Garzón et al. [14] and Yantir et al. [70] separately proposed a convolutional neural network design using an in-memory associative processor. Complete system designs of in-memory associative processors have been separately proposed by Zha and Li [71] and Caminal et al. [72]. Yantir [73] studied CMOS and resistive NOR CAM based associative processors and their applications.

To our knowledge, AM^4 is the first solution that converts a MRAM crossbar designed for random access storage, into an associative processor. We achieve that without altering the MRAM core, only by manipulating data and amending the peripheral circuitry.

VIII. Conclusion

In this work, we presented AM^4, a multiple purpose (i.e., CAM, TCAM, approximate CAM, and in-memory associative processor) NAND-type architecture based on the silicon-proven MTJ-based Samsung crossbar array. AM^4 enables a wide range of in-memory computing applications. We validated the basic AM^4 functionality and evaluated its timing and energy consumption by circuit-level simulations using Cadence EDA tools. AM^4 was designed using a commercial 28 nm FDSOI technology node. A Verilog-A based compact model for the DMTJ device was amended. Simulation results show that AM^4 may require data coding (such as ECC) to operate reliably due to very limited high resistance / low resistance ratio. We conducted an exhaustive design space exploration, showing that AM^4 exhibits very low susceptibility to process variations around even and skewed corners. AM^4 was applied to Smith-Waterman DNA sequence alignment, a frequent bioinformatics workload. AM^4 was shown to significantly outperform state-of-the-art conventional as well as other in-memory computing alternatives in terms of performance and energy-efficiency.

References

[1] R. Balasubramonian et al., "Near-data processing: Insights from a MICRO-46 workshop," *IEEE Micro*, vol. 34, no. 4, pp. 36–42, Jul./Aug. 2014.

[2] J. Ahn, S. Hong, S. Yoo, O. Mutlu, and K. Choi, "A scalable processing-in-memory accelerator for parallel graph processing," in *Proc. 42nd Annu. Int. Symp. Comput. Archit.*, Jun. 2015, pp. 105–117.

[3] J. Ahn, S. Yoo, O. Mutlu, and K. Choi, "PIM-enabled instructions: A low-overhead, locality-aware processing-in-memory architecture," in *Proc. 42nd Annu. Int. Symp. Comput. Archit.*, Jun. 2015, pp. 336–348.

[4] R. Nair et al., "Active memory cube: A processing-in-memory architecture for exascale systems," *IBM J. Res. Develop.*, vol. 59, nos. 2–3, pp. 1–17, 2015.

[5] L. Yavits, R. Kaplan, and R. Ginosar, "GIRAF: General purpose in-storage resistive associative framework," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 2, pp. 276–287, Feb. 2021.

[6] S. Jung et al., "A crossbar array of magnetoresistive memory devices for in-memory computing," *Nature*, vol. 601, no. 7892, pp. 211–216, Jan. 2022.

[7] M. Rakka, M. E. Fouda, R. Kanj, A. Eltawil, and F. J. Kurdahi, "Design exploration of sensing techniques in 2T-2R resistive ternary CAMs," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 68, no. 2, pp. 762–766, Feb. 2021.

[8] K. Zhou et al., "The trend of emerging non-volatile TCAM for parallel search and AI applications," *Chip*, vol. 1, Jun. 2022, Art. no. 100012.

[9] R. Kaplan, L. Yavits, R. Ginosar, and U. Weiser, "A resistive CAM processing-in-storage architecture for DNA sequence alignment," *IEEE Micro*, vol. 37, no. 4, pp. 20–28, Aug. 2017.

[10] K. Pan, A. M. S. Tosson, N. Wang, N. Y. Zhou, and L. Wei, "A novel 2T2R CR-based TCAM design for high-speed and energy-efficient applications," in *Proc. Great Lakes Symp. VLSI*, Jun. 2022, pp. 33–38.

[11] R. De Rose, M. d'Aquino, G. Finocchio, F. Crupi, M. Carpentieri, and M. Lanuzza, "Compact modeling of perpendicular STT-MTJs with double reference layers," *IEEE Trans. Nanotechnol.*, vol. 18, pp. 1063–1070, 2019.

[12] T. F. Smith and M. S. Waterman, "Identification of common molecular subsequences," *J. Molecular Biol.*, vol. 147, no. 1, pp. 195–197, 1981.

[13] L. Yavits, S. Kvatinsky, A. Morad, and R. Ginosar, "Resistive associative processor," *IEEE Comput. Archit. Lett.*, vol. 14, no. 2, pp. 148–151, Jul. 2015.

[14] E. Garzon, A. Teman, M. Lanuzza, and L. Yavits, "AIDA: Associative in-memory deep learning accelerator," *IEEE Micro*, vol. 42, no. 6, pp. 67–75, Nov. 2022.

[15] R. Kaplan, L. Yavits, and R. Ginosasr, "BioSEAL: In-memory biological sequence alignment accelerator for large-scale genomic data," in *Proc. 13th ACM Int. Syst. Storage Conf.*, May 2020, pp. 36–48.

[16] L. Yavits, A. Morad, and R. Ginosar, "Computer architecture with associative processor replacing last-level cache and SIMD accelerator," *IEEE Trans. Comput.*, vol. 64, no. 2, pp. 368–381, Feb. 2015.

[17] L. Yavits, A. Morad, and R. Ginosar, "Sparse matrix multiplication on an associative processor," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 11, pp. 3175–3183, Nov. 2015.

[18] R. Kaplan, L. Yavits, and R. Ginosar, "PRINS: Processing-in-storage acceleration of machine learning," *IEEE Trans. Nanotechnol.*, vol. 17, no. 5, pp. 889–896, Sep. 2018.

[19] Y. Ding, Y. Zhao, X. Shen, M. Musuvathi, and T. Mytkowicz, "Yinyang K-means: A drop-in replacement of the classic K-means with consistent speedup," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 579–587.

[20] Z. Li, J. Jin, and L. Wang, "High-performance K-means implementation based on a simplified map-reduce architecture," 2016, *arXiv:1610.05601*.

[21] N. Ramanathan, J. Wickerson, F. Winterstein, and G. A. Constantinides, "A case for work-stealing on FPGAs with OpenCL atomics," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, Feb. 2016, pp. 48–53.

[22] J. Bhimani, M. Leeser, and N. Mi, "Accelerating K-means clustering with parallel implementations and GPU computing," in *Proc. IEEE High Perform. Extreme Comput. Conf. (HPEC)*, Sep. 2015, pp. 1–6.

[23] C. J. Rossbach, Y. Yu, J. Currey, J.-P. Martin, and D. Fetterly, "Dandelion: A compiler and runtime for heterogeneous systems," in *Proc. 24th ACM Symp. Operating Syst. Princ.*, Nov. 2013, pp. 49–68.

[24] C. Eckert et al., "Neural cache: Bit-serial in-cache acceleration of deep neural networks," in *Proc. ACM/IEEE 45th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2018, pp. 383–396.

[25] G. Wang et al., "Compact modeling of perpendicular-magnetic-anisotropy double-barrier magnetic tunnel junction with enhanced thermal stability recording structure," *IEEE Trans. Electron Devices*, vol. 66, no. 5, pp. 2431–2436, May 2019.

[26] H. Sato, M. Yamanouchi, S. Ikeda, S. Fukami, F. Matsukura, and H. Ohno, "Perpendicular-anisotropy CoFeB-MgO magnetic tunnel junctions with a MgO/CoFeB/Ta/CoFeB/MgO recording structure," *Appl. Phys. Lett.*, vol. 101, no. 2, Jul. 2012, Art. no. 022414.

[27] S. Ikeda et al., "A perpendicular-anisotropy CoFeB–MgO magnetic tunnel junction," *Nature Mater.*, vol. 9, pp. 721–724, Sep. 2010.

[28] S. Jeloka, N. B. Akesh, D. Sylvester, and D. Blaauw, "A 28 nm configurable memory (TCAM/BCAM/SRAM) using push-rule 6T bit cell enabling logic-in-memory," *IEEE J. Solid-State Circuits*, vol. 51, no. 4, pp. 1009–1021, Apr. 2016.

[29] I. Arsovski and R. Wistort, "Self-referenced sense amplifier for across-chip-variation immune sensing in high-performance content-addressable memories," in *Proc. IEEE Custom Integr. Circuits Conf.*, Sep. 2006, pp. 453–456.

[30] D. H. Yoon, N. Muralimanohar, J. Chang, P. Ranganathan, N. P. Jouppi, and M. Erez, "FREE-p: Protecting non-volatile memory against both hard and soft errors," in *Proc. IEEE 17th Int. Symp. High Perform. Comput. Archit.*, Feb. 2011, pp. 466–477.

[31] K. Pagiamtzis and A. Sheikholeslami, "Content-addressable memory (CAM) circuits and architectures: A tutorial and survey," *IEEE J. Solid-State Circuits*, vol. 41, no. 3, pp. 712–727, Mar. 2006.

[32] R. Karam, R. Puri, S. Ghosh, and S. Bhunia, "Emerging trends in design and applications of memory-based computing and content-addressable memories," *Proc. IEEE*, vol. 103, no. 8, pp. 1311–1330, Aug. 2015.

[33] G. Simoni, F. Reali, C. Priami, and L. Marchetti, "Stochastic simulation algorithms for computational systems biology: Exact, approximate, and hybrid methods," *WIREs Syst. Biol. Med.*, vol. 11, no. 6, p. e1459, Nov. 2019.

[34] E. Garzon et al., "Hamming distance tolerant content-addressable memory (HD-CAM) for DNA classification," *IEEE Access*, vol. 10, pp. 28080–28093, 2022.

[35] R. Hanhan, E. Garzón, Z. Jahshan, A. Teman, M. Lanuzza, and L. Yavits, "EDAM: Edit distance tolerant approximate matching content addressable memory," in *Proc. 49th Annu. Int. Symp. Comput. Archit.*, Jun. 2022, pp. 495–507.

[36] E. Garzón, R. De Rose, F. Crupi, L. Trojman, A. Teman, and M. Lanuzza, "Relaxing non-volatility for energy-efficient DMTJ based cryogenic STT-MRAM," *Solid-State Electron.*, vol. 184, Oct. 2021, Art. no. 108090.

[37] Y. Liu, T.-T. Tran, F. Lauenroth, and B. Schmidt, "SWAPHI-LS: Smith-waterman algorithm on Xeon Phi coprocessors for long DNA sequences," in *Proc. IEEE Int. Conf. Cluster Comput. (CLUSTER)*, Sep. 2014, pp. 257–265.

[38] L. Wienbrandt, "The FPGA-based high-performance computer RIVYERA for applications in bioinformatics," in *Proc. Conf. Computability Eur.* Cham, Switzerland: Springer, 2014, pp. 383–392.

[39] E. F. D. O. Sandes, G. Miranda, X. Martorell, E. Ayguade, G. Teodoro, and A. C. M. Melo, "CUDAlign 4.0: Incremental speculative traceback for exact chromosome-wide alignment in GPU clusters," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 10, pp. 2838–2850, Oct. 2016.

[40] H. Lan, W. Liu, Y. Liu, and B. Schmidt, "SWhybrid: A hybrid-parallel framework for large-scale protein sequence database search," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, May 2017, pp. 42–51.

[41] NCBI. (2021). *Bethesda (MD): National Library of Medicine (US), National Center for Biotechnology Information*. [Online]. Available: https://www.ncbi.nlm.nih.gov/

[42] M. E. Fouda, H. E. Yantir, A. M. Eltawil, and F. Kurdahi, "In-memory associative processors: Tutorial, potential, and challenges," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 69, no. 6, pp. 2641–2647, Jun. 2022.

[43] A. T. Do, C. Yin, K. S. Yeo, and T. T.-H. Kim, "Design of a power-efficient CAM using automated background checking scheme for small match line swing," in *Proc. ESSCIRC*, Sep. 2013, pp. 209–212.

[44] D. Sethi, M. Kaur, and G. Singh, "Design and performance analysis of a CNFET-based TCAM cell with dual-chirality selection," *J. Comput. Electron.*, vol. 16, no. 1, pp. 106–114, Mar. 2017.

[45] D. Jothi and R. Sivakumar, "Design and analysis of power efficient binary content addressable memory (PEBCAM) core cells," *Circuits, Syst., Signal Process.*, vol. 37, no. 4, pp. 1422–1451, Apr. 2018.

[46] S. W. Hussain, T. V. Mahendra, S. Mishra, and A. Dandapat, "Match-line division and control to reduce power dissipation in content addressable memory," *IEEE Trans. Consum. Electron.*, vol. 64, no. 3, pp. 301–309, Aug. 2018.

[47] K. Prasanth et al., "High speed, low matchline voltage swing and search line activity TCAM cell array design in 14 nm FinFET technology," in *Emerging Trends in Electrical, Communications, and Information Technologies*. Cham, Switzerland: Springer, 2020, pp. 465–473.

[48] S. Mishra, T. V. Mahendra, and A. Dandapat, "A 9-T 833-MHz 1.72-fJ/bit/search quasi-static ternary fully associative cache tag with selective matchline evaluation for wire speed applications," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 63, no. 11, pp. 1910–1920, Nov. 2016.

[49] I. Arsovski et al., "1.4Gsearch/s 2-Mb/mm^2 TCAM using two-phase-pre-charge ML sensing and power-grid pre-conditioning to reduce Ldi/dt power-supply noise by 50%," *IEEE J. Solid-State Circuits*, vol. 53, no. 1, pp. 155–163, Jan. 2018.

[50] Y.-S. Chan et al., "0.4 V reconfigurable near-threshold TCAM in 28 nm high-K metal-gate CMOS process," in *Proc. 31st IEEE Int. Syst.-Chip Conf. (SOCC)*, Sep. 2018, pp. 272–277.

[51] Q. Dong et al., "A 4 + 2T SRAM for searching and in-memory computing with 0.3-V V_{DDmin}," *IEEE J. Solid-State Circuits*, vol. 53, no. 4, pp. 1006–1015, Apr. 2018.

[52] V. M. Zackriya and H. M. Kittur, "Precharge-free, low-power content-addressable memory," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 24, no. 8, pp. 2614–2621, Aug. 2016.

[53] M. A. Bahloul et al., "Design and analysis of 2T-2M ternary content addressable memories," in *Proc. IEEE 60th Int. Midwest Symp. Circuits Syst. (MWSCAS)*, Aug. 2017, pp. 1430–1433.

[54] A. K. Ramanathan et al., "Monolithic 3D+−IC based massively parallel compute-in-memory macro for accelerating database and machine learning primitives," in *IEDM Tech. Dig.*, Dec. 2020, p. 28.

[55] X. Zhu, X. Yang, C. Wu, J. Wu, and X. Yi, "Hamming network circuits based on CMOS/memristor hybrid design," *IEICE Electron. Exp.*, vol. 10, no. 12, 2013, Art. no. 130404.

[56] M. M. A. Taha and C. Teuscher, "Approximate memristive in-memory Hamming distance circuit," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 16, no. 2, pp. 1–14, Apr. 2020.

[57] K. Pagiamtzis, N. Azizi, and F. Najm, "A soft-error tolerant content-addressable memory (CAM) using an error-correcting-match scheme," in *Proc. IEEE Custom Integr. Circuits Conf.*, Sep. 2006, pp. 301–304.

[58] S. C. Krishnan, R. Panigrahy, and S. Parthasarathy, "Error-correcting codes for ternary content addressable memories," *IEEE Trans. Comput.*, vol. 58, no. 2, pp. 275–279, Feb. 2009.

[59] P. Reviriego, S. Pontarelli, and A. Ullah, "Error detection and correction in SRAM emulated TCAMs," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 2, pp. 486–490, Feb. 2019.

[60] I. Ullah, J.-S. Yang, and J. Chung, "ER-TCAM: A soft-error-resilient SRAM-based ternary content-addressable memory for FPGAs," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 4, pp. 1084–1088, Apr. 2020.

[61] T. T. Bui and T. Shibata, "A low-power associative processor with the R-th nearest-match Hamming-distance search engine employing time-domain techniques," in *Proc. 5th IEEE Int. Symp. Electron. Design, Test Appl.*, Jan. 2010, pp. 54–57.

[62] S. Amanollahi, M. Kamal, A. Afzali-Kusha, and M. Pedram, "Circuit-level techniques for logic and memory blocks in approximate computing systemsx," *Proc. IEEE*, vol. 108, no. 12, pp. 2150–2177, Dec. 2020.

[63] A. Rahimi, A. Ghofrani, K.-T. Cheng, L. Benini, and R. K. Gupta, "Approximate associative memristive memory for energy-efficient GPUs," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, 2015, pp. 1497–1502.

[64] Q. Guo, X. Guo, Y. Bai, and E. Ipek, "A resistive TCAM accelerator for data-intensive computing," in *Proc. 44th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Dec. 2011, pp. 339–350.

[65] H. E. Yantir, A. M. Eltawil, and F. J. Kurdahi, "A two-dimensional associative processor," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 26, no. 9, pp. 1659–1670, Sep. 2018.

[66] M. Hout, M. E. Fouda, R. Kanj, and A. M. Eltawil, "In-memory multi-valued associative processor," 2021, *arXiv:2110.09643*.

[67] M. Imani and T. Rosing, "CAP: Configurable resistive associative processor for near-data computing," in *Proc. 18th Int. Symp. Quality Electron. Design (ISQED)*, Mar. 2017, pp. 346–352.

[68] H. Caminal et al., "CAPE: A content-addressable processing engine," in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit. (HPCA)*, Feb. 2021, pp. 557–569.

[69] M. A. Neggaz, H. E. Yantir, S. Niar, A. Eltawil, and F. Kurdahi, "Rapid in-memory matrix multiplication using associative processor," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2018, pp. 985–990.

[70] H. E. Yantir, A. M. Eltawil, and K. N. Salama, "IMCA: An efficient in-memory convolution accelerator," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 29, no. 3, pp. 447–460, Mar. 2021.

[71] Y. Zha and J. Li, "Hyper-AP: Enhancing associative processing through a full-stack optimization," in *Proc. ACM/IEEE 47th Annu. Int. Symp. Comput. Archit. (ISCA)*, May 2020, pp. 846–859.

[72] H. Caminal, Y. Chronis, T. Wu, J. M. Patel, and J. F. Martínez, "Accelerating database analytic query workloads using an associative processor," in *Proc. 49th Annu. Int. Symp. Comput. Archit.*, Jun. 2022, pp. 623–637.

[73] H. E. Yantir, *Efficient Acceleration of Computation Using Associative in-Memory Processing*. Irvine, CA, USA: Univ. California, 2018.

Esteban Garzón (Member, IEEE) received the Ph.D. degree in electronics engineering from the University of Calabria (UNICAL), Italy, in 2022. He is currently a Post-Doctoral Researcher at the Department of Computer Engineering, Modeling, Electronics, and Systems Engineering, UNICAL. He has coauthored more than 30 scientific papers in international journals and conferences. He has participated in several IC tapeouts. His research interests include domain-specific hardware accelerators, electronics/spintronics, cryogenic memories, and standard and emerging technologies for logic, memory, and low-power applications.

Marco Lanuzza (Senior Member, IEEE) received the Ph.D. degree in electronic engineering from the Mediterranea University of Reggio Calabria, Reggio Calabria, Italy, in 2005. Since 2006, he has been with the University of Calabria, Rende, Italy, where he is currently an Associate Professor. He has authored more than 120 publications in international journals and conference proceedings. His current research interests include the design of ultralow voltage circuits and systems, development of efficient models and methodologies for leakage- and variability-aware designs, and the design of digital and analog circuits in emerging technologies. He is an Associate Editor of *Integration, the VLSI Journal.*

Adam Teman (Member, IEEE) received the Ph.D. degree in electrical and computer engineering from Ben-Gurion University (BGU), Be'er Sheva, in 2014. He worked as a Design Engineer at Marvell Semiconductors from 2006 to 2007. From 2014 to 2015, he was a Post-Doctoral Researcher at the École Polytechnique Fédérale de Lausanne (EPFL), Switzerland, under a Swiss Government Excellence Scholarship. Since 2015, he has been with Bar-Ilan University, where he is currently an Associate Professor and the Co-Director of Emerging Nanoscaled Integrated Circuits and Systems (EnICS) Laboratories. He has authored over 100 scientific articles and ten patents. His research interests include embedded memories, energy-efficient circuit design, hardware for artificial intelligence, open source processor platforms and accelerators, and methodologies for physical implementation. He is a member of the technical and review boards of several conferences and journals. In 2020, he was awarded the Krill Prize for Outstanding Young Researchers. He is an Associate Editor of the *Microelectronics Journal*

Leonid Yavits (Member, IEEE) received the M.Sc. and Ph.D. degrees in electrical engineering from Technion, Israel. He is currently with the Faculty of Engineering, Bar-Ilan University, Israel. He is also a serial entrepreneur, who was involved in co-founding and successful management (from a concept to M&A) of several start-ups in the field of ASICs. His research interests include bioinformatics, domain specific accelerators, and processing in memory.

Open Access funding provided by 'Università della Calabria' within the CRUI CARE Agreement