MnM: A Fast and Efficient Min/Max Searching in MRAM

Amitesh Sridharan*
amitesh.sridharan@asu.edu
School of Electrical, Computer and Energy Engineering,
Arizona State University
Tempe, Arizona, USA

Fan Zhang*
fzhang95@asu.edu
School of Electrical, Computer and Energy Engineering,
Arizona State University
Tempe, Arizona, USA

Deliang Fan
dfan@asu.edu
School of Electrical, Computer and Energy Engineering,
Arizona State University
Tempe, Arizona, USA

ABSTRACT

In-Memory Computing (IMC) technology has been considered to be a promising approach to solve well-known memory-wall challenge for data intensive applications. In this paper, we are the first to propose *MnM*, a novel IMC system with innovative architecture/circuit designs for fast and efficient Min/Max searching computation in emerging Spin-Orbit Torque Magnetic Random Access Memory (SOT-MRAM). Our proposed SOT-MRAM based in-memory logic circuits are specially optimized to perform parallel, one-cycle XNOR logic that are heavily used in the Min/Max searching-in-memory algorithm. Our novel in-memory XNOR circuit also has an overhead of just two transistors per row when compared to most prior methodologies which typically use multiple sense amplifiers or complex CMOS logic gates. We also design all other required peripheral circuits for implementing complete Min/Max searching-in-MRAM computation. Our cross-layer comprehensive experiments on Dijkstra's algorithm and other sorting algorithms in real word datasets show that our MnM could achieve significant performance improvement over CPUs, GPUs, and other competing IMC platforms based on RRAM/MRAM/DRAM.

CCS CONCEPTS

• **Hardware → Non-volatile memory**; **Emerging architectures**.

KEYWORDS

Min/Max, In-Memory-Computing, SOT-MRAM.

ACM Reference Format:
Amitesh Sridharan, Fan Zhang, and Deliang Fan. 2022. MnM: A Fast and Efficient Min/Max Searching in MRAM. In *Proceedings of the Great Lakes Symposium on VLSI 2022 (GLSVLSI '22), June 6–8, 2022, Irvine, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3526241.3530349

1 INTRODUCTION

Many data-centric problems, such as sorting, ranking, graph processing, data mining, bioinformatics, route planning. etc., have the fundamental operation of minimum or maximum (Min/Max) searching among bulk data [12, 16, 22]. Min/Max searching is also the critical and most time-consuming computation in a lot of popular algorithms, such as Dijkstra's algorithm to find the shortest path in a graph, Prim's and Kruskal's algorithm to find the minimum spanning tree, and some dynamic programming algorithms. Real-time social media and online applications also require fast Min/Max searching operations to rank up-to-the-minute information. Implementation of Min/Max searching in traditional Von-neuman architectures can be very inefficient, especially when dealing with big data processing. A well-known challenge called the 'memory-wall' poses long off-chip memory access latency due to limited bandwidth. Moreover, the energy consumed by such off-chip data movement is orders of magnitude higher than computation itself [7]. For traditional Min/Max searching operation, it is typically performed by comparing individual elements in the large data array so as to obtain the largest and smallest values, which inevitably requires frequent movement of data between different levels of memory hierarchy. This comparison of colossal data also results in continuous data movement between CPU and memory, which highlights Min/Max searching computation to be a victim of the memory-wall bottleneck.

In-Memory-Computing (IMC) was introduced as a promising candidate to solve the memory-wall issue. The IMC architectures focus on designing computing logic within the memory to minimize data movement and utilizing the minimized data movement during computation to significantly reduce overall system power consumption and improve performance. In particular, many non-volatile memory (NVM) technologies, like Resistive RAM (RRAM) or MRAM, are very promising candidates to pave a novel path to realize area and energy-efficient system supporting in-memory processing due to features like non-volatility, zero standby leakage, compatibility with CMOS fabrication process and excellent integration density [6, 18, 25]. Many IMC approaches based on different memory technologies, including SOT-MRAM, have been proposed to accelerate data-centric applications like machine learning [5, 13, 14], bio-informatics [25], graph processing[6], etc.

As one representative data-intensive application, it is natural to develop Min/Max searching implementation leveraging IMC architecture to minimize off-chip data access, thus significantly improving searching speed and reducing overall power consumption. As far as we know, the prior work, Max-PIM [24], first proposed 'Min/Max searching-in-memory' algorithm, where the core operation of this novel IMC-oriented min/max searching algorithm is the in-memory XNOR logic based bit-wise comparison for all the data stored in the same memory array. However, the in-memory logic design of Max-PIM comes with a large overhead of 3 sense

*Both authors contributed equally to this work.

GLSVLSI '22, June 6–8, 2022, Irvine, CA, USA

ACM ISBN 978-1-4503-9322-5/22/06…$15.00
https://doi.org/10.1145/3526241.3530349

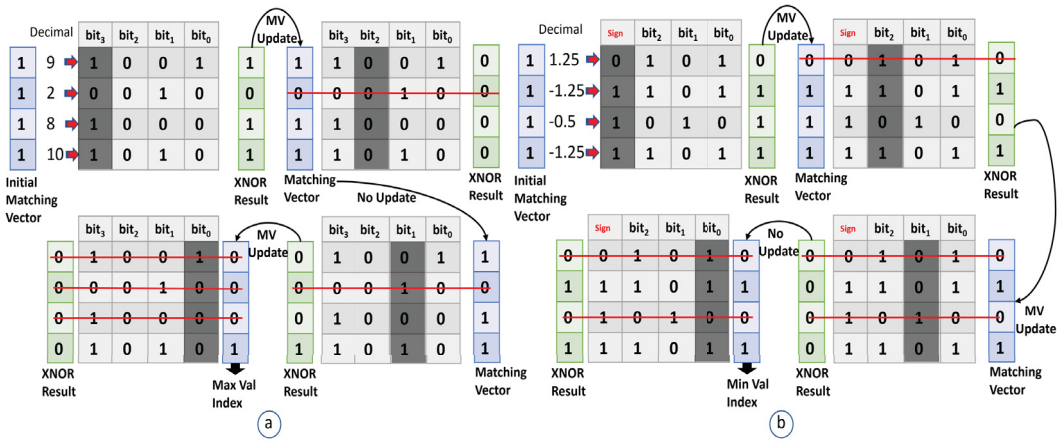


Figure 1: (a) Illustration of the parallel bit-wise XNOR operations to find maximum value. (b) Illustration of finding the minimum value for a signed fixed point number array.

amplifiers (SAs) per bit-line and a NAND logic gate to perform the single cycle XNOR logic. It also requires multiple row activation scheme which further increases power during compute, and has the inherent read disturb problem. Since Max-PIM is implemented in DRAM, the in-memory logic suffers from the intrinsic destructive read issue after logic evaluation. Thus, before searching operation, the target operand has to be copied to another memory array to avoid operand data destruction after computation, which inevitably occurs extra power, latency, and area consumption. In this work, Our proposed *MnM* works on alleviating all the aforementioned problems, which are summarized as follows:

(1) We are the first to propose, *MnM*, a fast and efficient Min/Max searching-in-memory architecture based on non-volatile Spin-Orbit Torque Magnetic Random Access Memory (SOT-MRAM), which supports in-memory searching for minimum and maximum out of bulk data stored within memory as unsigned or signed integers, fixed-point and floating numbers.

(2) The complete peripheral circuits of *MnM* are designed and verified to support all operations needed in Min/Max searching-in-memory algorithm. It is worth to note that our novel one-cycle in-memory XNOR logic design only comes with 2 transistors per bit-line overhead, which is the smallest as far as we know. The non-volatile SOT-MRAM based *MnM* could directly search min/max within the actual stored operand data array, without extra pre-processing of transposed data copy needed in DRAM based counterpart[24].

(3) We evaluate *MnM* on case studies of accelerating Dijkstra's algorithm [15] and other sorting algorithms using real world datasets [1]. Extensive cross-layer experiments are conducted to compare our *MnM* with other state-of-the-art IMC platforms (e.g., IMCE [3], Max-PIM [24], Pinatubo[18], and more), CPU and GPU, where our *MnM* shows substantial improvement in energy, speed, etc.

2 MIN/MAX SEARCHING-IN-MEMORY ALGORITHM

The Min/Max searching-in-memory algorithm proposed in [24] mainly performs parallel bit-wise XNOR based comparison for the data stored in one memory array. The algorithm finds the first/last ranked number in a given list of numbers. Its working mechanism mainly relies on the fact that different bit positions represent different significance, leveraging which it could gradually eliminate the smaller/larger number from MSB to LSB. To adapt for in-memory computing, it also fully leverages the parallel sensing of multiple bit lines to perform '*N*' parallel XNOR based bit-wise comparison in one single cycle, where *N* stands for the number of operands stored in memory. Algorithm-1 shows the pseudo code for Min/Max searching-in-Memory algorithm [24]. It has a constant searching time of $O(n)$, where n is the number of bits required to represent the operands. The critical part of the algorithm is that the bit-length of the operands are fixed for the aforementioned constant time comparisons. This avoids the inclusion of an early stop detection unit and greatly saves on hardware cost. The algorithm is compatible with unsigned, signed, fixed point numbers and with IEEE754 [23]

Algorithm 1: Parallel Bit-wise Min/Max-in-Memory

```
Input  : Array X has M elements, where each element contains N bits.
Result: Returning the Min/Max value in given array.
Storing the input array into 2D bit-array(NxM size) where each element in X occupies one
  column;
Matching_Vector = ones(1,M);
if find min then
    Comparison_Sign_Bit=1 ; ▷ For signed number, the sign bit and the rest
      have different XNOR operands
    Comparison_Bit = 0;
else
    Comparison_Sign_Bit=0;
    Comparison_Bit = 1;
end
if unsigned number then
    Comparison_Sign_Bit = Comparison_bit; ;     ▷ For unsigned bit, we do not
      need to distinguish sign bit and others.
end
while current_bit_position < N do
    ▷ Go through every bit from MSB to LSB.;
    if current_bit_position == 0 then
        Matching_Result = XNOR(current_bit,Comparison_Sign_Bit);
    else
        Matching_Result = XNOR(current_bit,Comparison_Bit);
    end
    if Matching_Result == All_Zeros then
        Continue;
    else
        Matching_Vector = Matching_Result;
    end
end
```

floating point numbers. The Fig. 1 (a) explains the Max searching-in-Memory algorithm with unsigned integers and Fig. 1 (b) explains the Min searching-in-Memory algorithm with signed fixed point numbers. The operands in this example are 4-bits in length, hence four iterations are required to complete the searching. The algorithm has three basic steps:

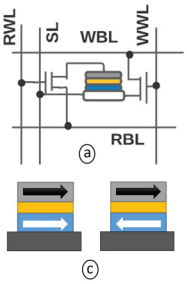
Step 1: Matching Vector(MV) Initialization.
Step 2: Parallel bit-wise comparison with comparison bit (CB) of all remaining operands selected by current MV.
Step 3: MV update.

This is followed by recursive call of the 2nd and 3rd steps based on the updated MV until bit-length is reached. To explain it further using unsigned integer as an example, for Max searching-in-memory, the algorithm first initializes the matching vector (MV) with all '1's, whose length equals to the number of operands to be compared, limited by the number of rows per memory array. For the first iteration, the parallel XNOR logic is conducted between the most significant bit (MSB) of the operands and an all-one Comparison vector(CV). If all the XNOR outputs from the first iteration are zeros, then the MV isn't changed and XNOR operations are performed between the next significant bits and the all-one CV again. If some of the XNOR based comparison outputs are '1s' in the first iteration, it means that the MSBs of those operands are '1'. Correspondingly, those identified numbers are comparatively larger numbers, while the other numbers could be excluded/eliminated for next bit comparison. Hence the MV which is updated every iteration based on the XNOR outputs, keeps track of this elimination and works as a control signal for the numbers that should be compared for the recursive call of the bit-wise comparison in the next iteration. The updated MV[i] with '1' indicates the i^{th} location will be compared in next iteration, and vice versa. It is repeated until LSB of the operands is reached. For min searching-in-memory operations, the overall procedure is similar to the max operation but with a small modification. Instead of performing XNOR operations with an all-one comparison vector as a start, an all-zero comparison vector is used. As for Max(or min) in signed integers, the algorithm first checks the sign bit (MSB), if a negative value(or positive value) is found, it will exclude all of those for the subsequent iteration. The approach is similar for fixed point numbers as well. For the IEEE754 floating point numbers, it is represented in three components: signed bit, exponent bits and significand bits. The bits in the exponent position carry a greater significance than the bits in the significand position which emphasize the precision. Hence this is similar to unsigned integers where the bit position dictates the bit significance. So this algorithm can be applied without any modification to comply with the IEEE754 floating point numbers.

3 PROPOSED *MNM* PLATFORM

3.1 Spin Orbit Torque MRAMs

Fig. 2a depicts the SOT-MRAM bit cell. It consists of two access transistors and the SOT-MTJ (Magnetic Tunnel Junction) [10, 25]. The SOT-MTJ is comprised of the Spin Hall heavy metal (SHM) and MTJ. The MTJ typically consists of two ferromagnetic layers with a tunnel barrier in between them. The Tunnel Magneto Resistance effect results in device to be in two different states, a low resistance state when the magnetization of ferromagnetic layers are parallel to



Operations	Write "1"("0")	Read
WWL	VDD	0
RWL	0	VDD
RBL	0	I_{sense}
WBL	$V_{WP}(V_{WN})$	0
SL	0	0

b

Figure 2: (a) SOT-MRAM Bit-Cell (b) Operation table for the SOT-MRAM (c) Parallel State and Anti-Parallel State

each other, and a high resistance state when they are anti-parallel to each other as shown in Fig 2c. The binary '0/1' is encoded as these low/high resistance states. The two access transistors are used per cell to provide write/read current to the SHM/MTJ for data write/read, respectively. Each bit-cell has the Write Word Line(WWL), Write Bit Line(WBL), Source Line(SL), Read Word Line(RWL) and Read Bit line(RBL). The biasing conditions of the SOT-MRAM are provided in the operation table in Fig. 2b.

3.2 Data Organization and Structuring of Memory Array

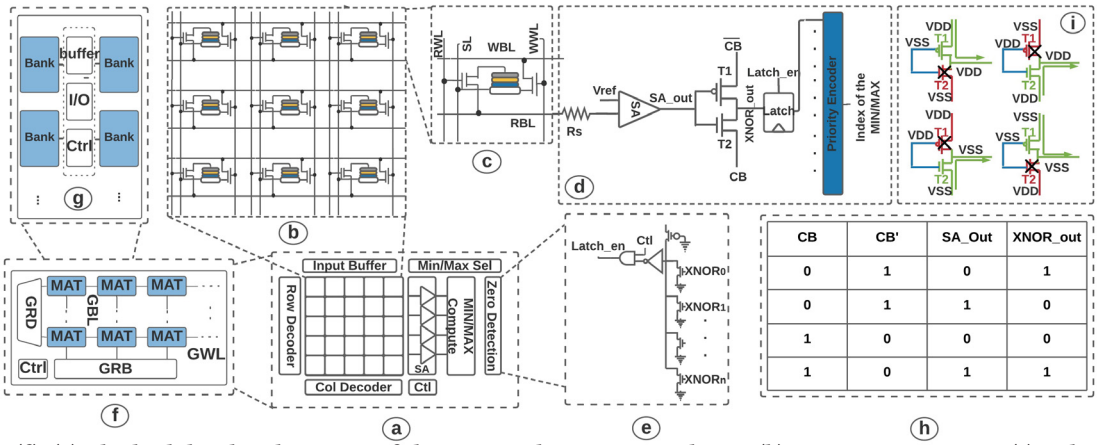
Our proposed *MnM* is designed to be an independent high-performance, parallel, and energy-efficient accelerator based on main memory architecture. The hierarchy structure is given in Fig.3. It contains multiple banks, sharing I/O, buffer and control units. Each bank is divided to multiple MATs connected to a Global Row Decoder (GRD) and a shared Global Row Buffer (GRB). Each MAT consists of 2D arrays of computational SOT-MRAM arrays. As discussed in prior work [24], the Min/Max searching-in-memory algorithm requires a transposed memory array design since it requires parallel logic between the same significant bit for different data across different memory rows. Following the same design requirement, our MnM also implements a transposed memory array design to attach each sense amplifier (SA) in row-wise fashion. For simplicity, we still call it bit-line even if it is in row wise direction as shown in Fig. 2 (a) and Fig. 3(c). Thus, the write/read word lines are in column-wise direction. Note that, one operand data is still stored in one row. As shown in Fig.3(a), a column decoder is used to select the N^{th} bit of the operands and the row decoder is capable of selecting all the rows at once for parallel in-memory logic.

3.3 Circuit and Architecture

Based on min/max searching-in-memory algorithm discussed in Section II, to implement the required computation, we designed the MnM architecture and complete peripheral circuits as shown in Fig. 3, with main functionalities:

(1) **Parallel bit-wise one-cycle in-memory XNOR.**
(2) **Matching vector update.**
(3) **All zero detection unit.**
(4) **Min/Max address decode.**

After the matching vector (MV) and comparison vector (CV) initialization, the searching starts from the first column (RWL⟨0⟩). It selects the MSBs of all the operands stored in the memory. Then,



CB	CB'	SA_Out	XNOR_out
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1

Figure 3: (a), (f), (g) The high level architecture of the proposed MRAM IMC design (b) Array Organization. (c) Enlarged Bit-Cell orientation, (d) Min/Max circuity, (e) All zero detection unit. (h) 2T-XNOR Truth Table. (i) in-memory 2T-XNOR logic circuit.

our proposed parallel in-memory XNOR logic circuit (called 2T-XNOR shown in Fig. 3 (d)) could directly implement the comparison between all the MSBs and the all-zero/one CV based on min/max selection. The first input to the 2T-XNOR is the SA_out which is the read-out from the memory array, the second input is the corresponding comparison bit, (CB, either 0 or 1 controlled by Min/Max selector) a subset of CV. The Min/Max selector ties the inputs to VSS or VDD based on the type of operation(Min/Max) and number format (unsigned, signed, fixed point and floating point). The detailed circuit, truth table, and how the proposed in-memory XNOR works are described in the Fig. 3 (d), (h) and (i) respectively. One of the inputs (represented as Comparison Bit(CB)) is given to the source terminal of PMOS **T1** and its complement is given to the source terminal of the NMOS **T2**. The other input (SA_Out, which is the to-be-compared bit value sensed through attached SA) is given to the gate terminal of the PMOS and NMOS. When the inputs are both digital 0's(VSS), the transistor **T1** is in the ON condition and **T2** is in the OFF condition. XNOR_out node is a copy of the source terminal of the ON transistor, hence CB' is passed to the XNOR_out node which results in the output being a digital 1. Now, when the input is '11', T1 is OFF and T2 is ON, input CB is mapped to the XNOR_out node which results in the output being a digital 1(VDD). Similarly, when the inputs - SA_out and CB are complementary, the output is 0. This exactly implements the XNOR logic operation, which only requires 2 transistors overhead after a normal memory SA. Comparing with prior in-memory XNOR logic designs [18, 24, 25], where they either use multiple cycles or need 1/2 extra SA(s) combined with complex CMOS logic designs, our proposed in-memory XNOR logic only incurs 2 extra transistors to implement in-memory XNOR logic within only one sensing cycle. Moreover, unlike many other in-memory logic designs suffering from reduced sensing margin since they either leverage the charge sharing in DRAM or sensing multiple cell combined resistance value in NVM, our in-memory XNOR logic does not have reduced sensing margin and lower power since it only needs to sense one cell and implement the logic in digital domain. As explained in Section II, after the parallel bit-wise in-memory XNOR based comparison, certain operands will be eliminated for next round of comparison through updating the corresponding MV value, except for all-zero XNOR outputs. Thus, for every round, an 'all-zero detection circuit' is needed . As shown in Fig. 3(e), it is implemented based on pseudo OR gate, that could detect the all-zero case, which controls the latch enable that stores the MV. Thus, only during the all zero case, the latch is disabled to prevent the MV update. When all the iterations of the XNOR based comparisons are done, the final MV stored in the latches denote the index(indices) of the Min/Max value(s). We leverage a priority encoder to return the identified min/max data address, which is able to handle the case when there are more than one minimum/maximum values stored in the same array. When all the array-level min/max values are achieved, they will be sent to higher level for further comparison based on similar mechanism.

4 EVALUATION AND RESULTS

4.1 Experiment Setup

The experimental setup used in this work is similar to [11, 24]. The SOT-MRAM model and device parameters are obtained from [11]. To analyze the performance of our IMC platform, we use the 1024x256 memory array size and peripheral circuits are implemented in the 45nm NCSU Process Development Kit. Cadence Spectre is used to carry out the simulations to obtain the circuit-level performance metrics. The architectural-level metrics are obtained from a simulator based on the NVSim [9] and NVSim-CAM[2]. Further detailed experiments are conducted to compare the performance metrics among popular IMC platforms .

4.2 Circuit Simulations

Fig. 4 depicts the transient simulations comprising of all the cases for in-memory XNOR logic to validate our proposed design. It is assumed in the first couple of cycles that all the data are written to the 1024x256 sized MRAM array and is ready for the computation stage. The experiment setup in Fig. 4 shows the reads from two bit-cells denoted by subscripts 0 and 1. The bit-cell$_0$ has the value 0 stored in it and the bit-cell$_1$ has the value 1 stored in it. This is depicted in Fig. 4 SA_out$_{0/1}$. For the Min operation, one of the XNOR operand is 0. Therefore the CB is tied to VSS. Similarly for the Max operation the CB is tied to VDD. The sensing voltage

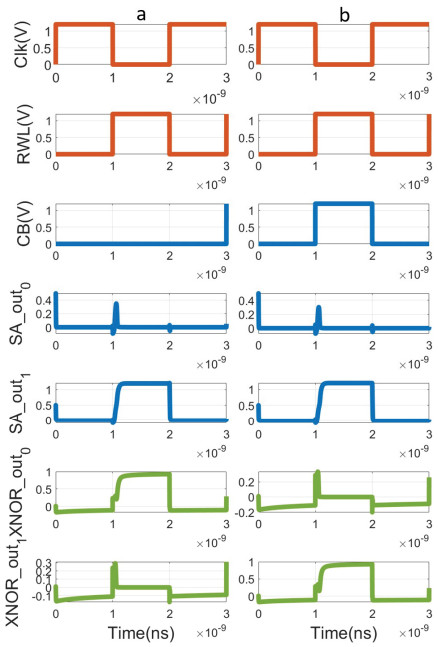


Figure 4: (a)Transient Simulations for the 2T XNOR with one input tied to '0' (b)Transient Simulations for the 2T XNOR with one input tied to '1'

V_{sense} is generated through a voltage divider circuit that divides the voltage between the $R_s = 5k\Omega$ and the resistance of the MRAM bit-cell. The formula for which is as follows : $V_{sense} \simeq \frac{R_{M1}}{R_{M1}+R_S} \times V_{read}$. Where R_{M1} is the $R_{Anti\text{-}Parallel}/R_{Parallel}$ of the Magnetic Tunnel Junction(MTJ) and V_{read} is the read voltage applied to the RBL.

4.3 Experiment Results

Since this is the first work to demonstrate min/max searching-in-MRAM, several non-volatile memory based IMC designs which have in-memory XNOR operations at its core are used for comparison. We draw our comparisons to Pinatubo[18], PIM-Quantifier[25], IMCE[3], some CAM based designs [20],[21],[17], and the DRAM based design Max-PIM[24].

In our experiments, we set each bank to have 16 MATs and each MAT consists of 2 sub-arrays with size of 1024×256. Thus, one bank stores ~1MB data. Considering 1GB memory capacity, there are 1024 banks for the whole memory. As the competitors, we built those platforms with the same organization as MnM, but with their own in-Memory computing logic circuits. For performance evaluation, we use the similar cross-layer evaluation framework in[4], starting from circuit-level evaluation with NCSU 45nm technology. We use the same process node to re-implement all counterpart designs. The memory peripherals are simulated using the same technology library in Synopsys Design Compiler. We then feed the circuit simulation results into architecture-level tools. We extensively modify NVSim[9] and NVSim-CAM[2]to extract the performance results. An in-house mapping algorithm is then developed on top of architecture level to parse the input vector coming from various data-sets and evaluate the performance. To evaluate the performance of our *MnM* platform for Min/Max searching in the real world dataset, we adopt a popular data-set T10I4D100K[1] containing 1010228 numbers, where each number is represented as 256-bit unsigned integer numbers stored in memory. We report the execution time and energy consumption measurement in Table.2 for different computing platforms including PIM-Quantifier, IMCE, Pinatubo, CPU, GPU, etc. For all IMC platforms, including our *MnM*, PIM-Quantifier, IMCE, and Pinatubo, such a large array cannot fit into a single computational array, requiring necessary data partitioning. The total 1010228 numbers are partitioned into 987 computational sub-arrays, where each sub-array will compute in parallel to return a local minimal number. Then, such local minimal numbers will be written into another computational array to get the global minimal number. For CPU&GPU evaluation, the reported time is total execution time, including data loading from main memory and processing. For energy estimation, similar as prior work[19, 24], we scale down 50% of CPU&GPU average power to exclude the power cost by cooling, voltage regulators, etc. As shown in Table 1 and 2, among other IMC platforms, *MnM* achieves minimal latency and energy consumption. It also has the least overhead in implementing the in-memory XNOR logic. Although by leveraging the DRAM, the Max-PIM has shorter latency and read/write energy per row. In terms of the XNOR latency/energy, our MnM outperformed the DRAM-based Max-PIM. That is because of the inherent destructive read property of DRAM, Max-PIM needs to write a copy of the data into another memory array before performing the XNOR operation. This transfer writing not only increases the latency and energy but also wastes the precious area. Overall, our *MnM* platform could achieve over 1000× speed improvement, and at least one order smaller energy consumption than GPUs/CPUs.

4.4 Applications in Graph Processing

In this section, we evaluate our *MnM* platform's performance with real-world application using Dijkstra's algorithm in graph processing. Dijkstra's algorithm [15] is a popular and widely used algorithm in graph processing to find the shortest path in a large graph. In Dijkstra's algorithm, Min/Max searching dominates overall computation. A breakdown of the overall computation in Dijkstra's algorithm is shown in Fig. 5. To evaluate it, three different dataset are used here: geom, foldoc and EAT_SR [8]. Table 3 reports the Min/Max searching speed improvement over CPU for different in-memory computing platforms. It can be seen that all in-memory computing platforms outperform software implementation in CPU by three orders mainly due to saving large amount of off-chip data movement and ultra-parallel processing capability. Aligning with prior experiments in Min/Max searching only case, our *MnM* for Dijkstra algorithm achieves the best performance compared with other in-memory computing platforms due to its optimized efficient XNOR and peripheral circuits. In general, in-memory computing platform could all achieve two to three orders speed up over CPU and our *MnM* still outperforms all other in-memory computing platforms significantly.

Table 1: Energy and Latency of different platforms

	Volatile Memory	Non-volatile Memory Design						
	Max-PIM (DRAM)[24]	MnM (This work)	PIM-Quantifier[25]	IMCE[3]	Pinatubo[18]	RRAM CAM[20]	MRAM CAM[21]	PCM CAM[17]
Technology(nm)	65nm	45nm	45nm	45nm	45nm	45nm	45nm	45nm
Latency(ns)	649ps(read) 649ps(write)	2.56ns(read) 1.77ns(write)	3.69ns(read) 1.66ns(write)	3.691ns 1.840ns	6.994ns 5.968ns	7.79ns 17.76ns	150.61ns 32.59ns	30.69ns 100ns
Energy(Read/row pJ)	33.2pJ	37.55pJ	90.94pJ	135.940pJ	137.436pJ	54.43pJ	697.28pJ	116.7pJ
Energy(Write/row pJ)	1.5pJ	60.77pJ	61.34pJ	92.092pJ	1.088nJ	1.200nJ	147.96pJ	7.34nJ
XNOR Latency(ns)	3.3ns	2.56ns	3.69ns	3.691ns	6.994ns	7.79ns	150.61ns	30.69ns
XNOR Energy(per row pJ)	71.6pJ	37.55pJ	90.94pJ	135.94pJ	137.436pJ	54.43pJ	697.28pJ	116.7pJ
XNOR area(No. of transistors)	49T/Col	17T/Row	40T/Col	18T+3R/Col	19T+3C+3R/Col	4T2R/Bit	4T2MTJ/Bit	2T2R/Bit

Table 2: Min/Max searching performance comparison

	Non-volatile Memory				Volatile Memory		
	MnM	PIM-quantifier	IMCE	Pinatubo	Max-PIM	CPU	GPU
Time:	1.31uS	1.89uS	1.89uS	3.58uS	1.69uS	8.6mS	684uS
Energy	9.8uJ	23.8uJ	35.6uJ	36uJ	18.77uJ	86mJ	18mJ

Table 3: Min/Max searching speedup over CPU in Dijkstra's Algorithm

	geom(7343 nodes)	foldoc(13356)	EAT_SR(23219)
Ours	905X	1652X	4579X
PIM-quantifier	627X	1145X	3173X
IMCE	627X	1145X	3173X
Pinatubo	330X	604X	1674X

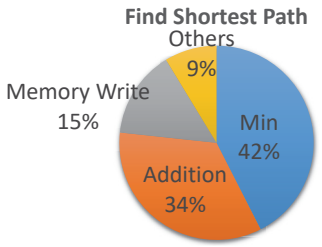


Figure 5: Dijkstra's Algorithm Analysis

5 CONCLUSION

In this work, we propose a novel SOT-MRAM based in-memory computing platform which supports the Min/Max searching-in-memory algorithm and performs it in a highly efficient manner. Compared to other NVM/DRAM based Processing-in-memory designs, our design outperforms all others by a significant margin for similar tasks.

ACKNOWLEDGMENTS

This work is supported in part by the National Science Foundation under Grant No.2003749 and No. 2144751.

REFERENCES

[1] [n.d.]. *T10I4D100K dataset.* http://fimi.cs.helsinki.fi
[2] S. Li andothers. 2016. NVSim-CAM: A circuit-level simulator for emerging nonvolatile memory based Content-Addressable Memory. In *2016 ICCAD*. 1–7.
[3] S. Angizi et al. 2018. IMCE: Energy-efficient bit-wise in-memory convolution engine for deep neural network. In *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*. 111–116.
[4] Shaahin Angizi et al. 2019. Redram: A reconfigurable processing-in-dram platform for accelerating bulk bit-wise operations. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*.
[5] Shaahin Angizi, Zhezhi He, Adnan Siraj Rakin, and Deliang Fan. 2018. CMP-PIM: An Energy-Efficient Comparator-based Processing-In-Memory Neural Network Accelerator. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. 1–6. https://doi.org/10.1109/DAC.2018.8465860
[6] Shaahin Angizi, Jiao Sun, Wei Zhang, and Deliang Fan. 2019. GraphS: A graph processing accelerator leveraging SOT-MRAM. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 378–383.
[7] Amirali Boroumand et al. 2018. Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks. *SIGPLAN Not.* 53, 2 (March 2018), 316–331.
[8] Timothy A. Davis and Yifan Hu. 2011. The University of Florida Sparse Matrix Collection. *ACM Trans. Math. Softw.* 38, 1, Article 1 (Dec. 2011), 25 pages. https://doi.org/10.1145/2049662.2049663
[9] Xiangyu Dong et al. 2012. Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory. *IEEE TCAD* 31 (2012), 994–1007.
[10] Xuanyao Fong, Yusung Kim, Karthik Yogendra, Deliang Fan, Abhronil Sengupta, Anand Raghunathan, and Kaushik Roy. 2015. Spin-transfer torque devices for logic and memory: Prospects and perspectives. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 35, 1 (2015), 1–22.
[11] Zhezhi He, Shaahin Angizi, Farhana Parveen, and Deliang Fan. 2017. High performance and energy-efficient in-memory computing architecture based on SOT-MRAM. In *2017 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*. 97–102. https://doi.org/10.1109/NANOARCH.2017.8053725
[12] Mohamed A Ismail et al. 1989. Multidimensional data clustering utilizing hybrid search strategies. *Pattern recognition* 22 (1989), 75–89.
[13] Zhewei Jiang, Shihui Yin, Jae-Sun Seo, and Mingoo Seok. 2020. C3SRAM: An In-Memory-Computing SRAM Macro Based on Robust Capacitive Coupling Computing Mechanism. *IEEE Journal of Solid-State Circuits* 55, 7 (2020), 1888–1897. https://doi.org/10.1109/JSSC.2020.2992886
[14] Zhewei Jiang, Shihui Yin, Mingoo Seok, and Jae-sun Seo. 2018. XNOR-SRAM: In-Memory Computing SRAM Macro for Binary/Ternary Deep Neural Networks. In *2018 IEEE Symposium on VLSI Technology*. 173–174. https://doi.org/10.1109/VLSIT.2018.8510687
[15] Donald B. Johnson. 1973. A Note on Dijkstra's Shortest Path Algorithm. *J. ACM* 20, 3 (July 1973), 385–388. https://doi.org/10.1145/321765.321768
[16] Roger V Lebo. 1982. Chromosome sorting and DNA sequence localization. *Cytometry: The Journal of the International Society for Analytical Cytology* 3 (1982), 145–154.
[17] J. Li et al. 2014. 1 Mb 0.41 μm² 2T-2R Cell Nonvolatile TCAM With Two-Bit Encoding and Clocked Self-Referenced Sensing. *IEEE Journal of Solid-State Circuits* 49, 4 (2014), 896–907. https://doi.org/10.1109/JSSC.2013.2292055
[18] S. Li et al. 2016. Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories. In *53rd DAC*. 1–6.
[19] S. Li et al. 2017. DRISA: A DRAM-based Reconfigurable In-Situ Accelerator. In *2017 MICRO*. 288–301.
[20] Li-Yue Huang et al. 2014. ReRAM-based 4T2R nonvolatile TCAM with 7x NVM-stress reduction, and 4x improvement in speed-wordlength-capacity for normally-off instant-on filter-based search engines used in big-data processing. In *Symposium on VLSI Circuits Digest of Technical Papers*. 1–2. https://doi.org/10.1109/VLSIC.2014.6858404
[21] S. Matsunaga et al. 2012. A 3.14 um2 4T-2MTJ-cell fully parallel TCAM based on nonvolatile logic-in-memory architecture. In *VLSIC*.
[22] Lawrence Page et al. 1999. *The PageRank citation ranking: Bringing order to the web.* Technical Report. Stanford InfoLab.
[23] Wikipedia contributors. 2020. IEEE 754 — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=IEEE_754&oldid=976446634. [Online; accessed 5-September-2020].
[24] Fan Zhang et al. 2021. Max-PIM: Fast and Efficient Max/Min Searching in DRAM. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*. 211–216. https://doi.org/10.1109/DAC18074.2021.9586096
[25] Fan Zhang et al. 2021. PIM-Quantifier: A Processing-in-Memory Platform for mRNA Quantification. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*. 43–48. https://doi.org/10.1109/DAC18074.2021.9586144