MPIM: Multi-Purpose In-Memory Processing Using Configurable Resistive Memory

Mohsen Imani, Yeseong Kim, Tajana Rosing

Computer Science and Engineering, UC San Diego, La Jolla, CA 92093, USA
{moimani, yek048, Tajana}@ucsd.edu

Abstract - Running Internet of Things applications on general purpose processors results in a large energy and performance overhead, due to the high cost of data movement. Processing in-memory is a promising solution to reduce the data movement cost by processing the data locally inside the memory. In this paper, we design a Multi-Purpose In-Memory Processing (MPIM) system, which can be used as main memory and for processing. MPIM consists of multiple crossbar memories with the capability of efficient in-memory computations. Instead of transferring the large dataset to the processors, MPIM provides two important in-memory processing capabilities: i) data searching for the nearest neighbor ii) bitwise operations including *OR*, *AND* and *XOR* with small analog sense amplifiers. The experimental results show that the MPIM can achieve up to 5.5x energy savings and 19x speedup for the search operations as compared to AMD GPU-based implementation. For bitwise vector processing, we present 11000x energy improvements with 62x speedup over the SIMD-based computation, while outperforming other state-of-the-art in-memory processing techniques.

I. Introduction

Internet of Things (IoT) designs are moving toward more data-driven and autonomous information retrieving solutions. The rate of data generation and the size of application datasets are anticipated to increase significantly [1-3] . A fundamental requirement for efficient IoT hardware designs is high performance and energy efficiency for large scale data computations.

There is a large body of research on processor architectures for efficient data processing. A popular solution is to leverage parallel architectures that mask the computation burden by taking advantage of multiple general purpose processors [4]. However, even using the massively many core systems consisting of CPUs and GPUs, the current system architecture cannot efficiently process large datasets. This inefficiency is the inevitable consequence of the large amount of data movements across the memory hierarchy due to small cache capacity and limited memory bandwidth. For example, one of the critical operations in IoT applications is fast search through a dataset and classification of data via search [5]. Other applications for graph processing [6], digital signal processing [7], communication [8] perform frequent bitwise computations for fetched memory data.

Processing in-memory (PIM) has been considered as a promising solution which could overcome the inefficiency in today's systems [9-12]. Instead of sending a large amount of data to the processing units for computation, PIM-based memories perform the computation inside the memory, thus the application performance can be accelerated by avoiding the memory access bottleneck [13]. However, integrating memory and logic on same die is not cost effective since high density memories and high performance logic require different design rules. 3D stacking has recently opened new opportunities in this area since memory can easily interact with computational logic [14]. However, this technique requires massive through-silicon-vias to connect logic to multi-layer memory stack [14]. For example, the work published in [15],[16] enables memory-based computation using large digital peripheral circuitries.

In this paper, we propose a new resistive memory design, called MPIM, which supports multiple in-memory processing operations in addition to traditional memory functionality. We design a cost-efficient PIM by utilizing analog characteristics of emerging non-volatile memory (NVM) technology. We address two important requirements to process the large amount of data stored in memory rows on our high-density crossbar memristor devices integraded in 3D on top of DRAM. First, we support a fast search operation to find the data of interest. Second, MPIM also supports bitwise operations even for multiple operands stored in different rows. When MPIM is configured to optimize search, it performs row-parallel search based on the timing difference of discharging current over the number of mismatched bits. For bitwise computation, we exploit an analog sense amplifier which can act as an *AND, OR*, and *XOR* gate. We implemented the proposed MPIM design using HSPICE simulator, and compare the energy and performance to recent processor architectures and state of the art PIM designs [16, 17]. Our experimental results show that running k-nearest neighbor (k-NN) search operation can improve energy and performance by 5.5**x** and 19**x** respectively compared to GPU-based k-NN running on AMD Southern Island GPU. MPIM can achieve 11000**x** energy efficiency improvement and 62**x** speedup for bitwise operations compared to GPU-based SIMD machine. We also show that our design also outperforms existing state-of-the-art in-memory processing techniques [16, 17] by 1.8**x** and 6.7**x** in terms of the energy and performance respectively. All these advantages are achieved with less than 5.1% area overhead compared to conventional NVM with just memory support.

II. Related Work

Processing in-memory can accelerate computation by reducing the overhead of data movement [18-20]. Early PIM designs integrate high performance CMOS logic and memory

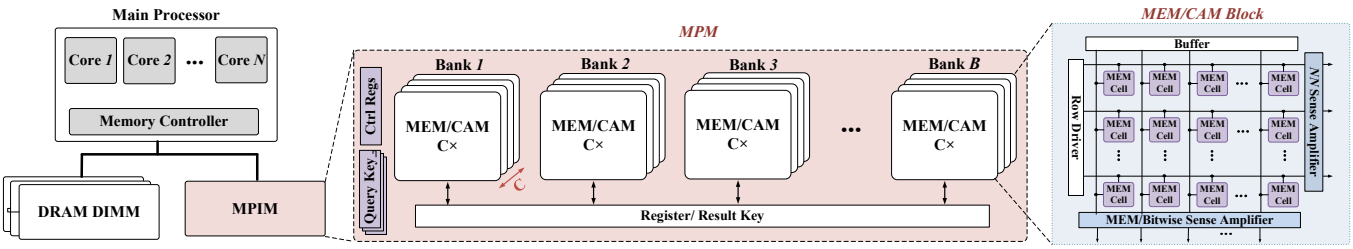


Figure 1. The overview of MPIM architecture.

on the same die. For example, the associative memory performs memory-based computation using content addressable memories (CAM) [21]. In CMOS technology, CAMs are designed with SRAM cells to provide high performance at significant energy cost for each search operations. Work in [20] also enables in-memory processing for some instruction types without changing the existing sequential programing model. The work in [5] designed a custom processing unit next to DRAM sense amplifiers to support slow row-serial search operation. However, to process other computing operations, processor cores are added as dedicated building blocks, making the manufacturing process complicated and costly.

The emerging NVM technology such as phase change memory, spin-transfer torque magnetic random access memory, and resistive RAM are good candidates to design memory [22-27], or enable PIM [15, 20, 28] due to its high density, low-power consumption, and scalability. For example, work in [19] presents a bit-serial TCAM design which addresses the limited flexibility of NVM-based TCAMs. Work in [17] exploits analog characteristic on NVMs to perform bitwise computations with two rows as its operands. These techniques perform in-memory computing for relatively small data sizes. One way to handle larger datasets is to support scalable computational operations as well as search functionality inside the TCAM. Our proposed design enables multiple in-memory operations, which are designed to process a large amount of data, in addition to original memory functionality. For example, the supported in-memory computations can process bulk OR operations for more than 200 rows which is ~2x faster than state-of-the-art PIMs.

III. Multi-Purpose In-Memory Processing

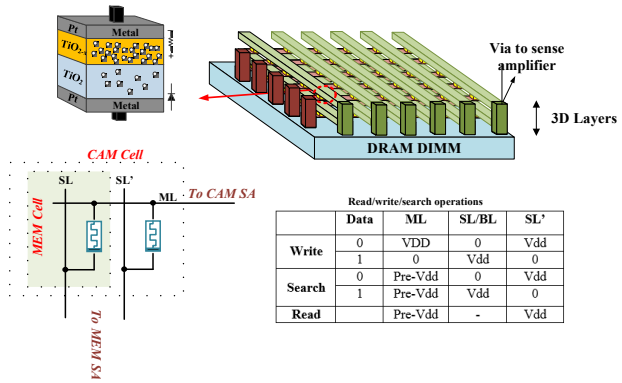
In this paper, we propose a multi-purpose resistive memory, called MPIM. Table 1 summarizes three key functionalities of MPIM with their application domains: i) load/store operations as an additional memory next to DRAM, ii) nearest neighbor search operation and iii) bitwise computation. First, as a memory, MPIM is excellent for more read intensive data which is often used for in-memory computation, while other frequently-updated data can be kept in DRAM. Second, MPIM supports a block-serial row parallel search operation that can be used as a building block for several applications which require this type of search. MPIM also supports bulk bitwise computations including *OR, AND, XOR*. This capability allows us to use MPIM to process general streaming applications such as multimedia and graph processing.

Table 1. MPIM operations and application examples

MPIM Operation	Applications
Memory	Non-write intensive workloads, keeping reference data
Bitwise computation	Stream processing, Multimedia, Graph processing, Digital signal processing, communication/security
Nearest neighbor search	Machine learning, Clustering, Statistical classifications, Database, Pattern recognition, Computer vision, Coding, Data compression

Figure 1 shows an architecture of the proposed MPIM. It consists of multiple memory banks which play a role of both memory and processing units. As a memory, MPIM can store and load general data. Stored data can be used for further processing. For data which will be used for in-memory processing, the main processor can request the data to be directly fetched from the hard disk to MPIM instead of DRAM. MPIM includes ***B*** banks, where each bank contains ***C*** crossbar memory which can be configured as either a memory mode for bit-wise computation, or in CAM mode for search.

Figure 2 illustrates the detail of the MPIM memory block. It is based on crossbar memristor devices, since they are high density and cost-effective replacement of main memory [29]. The crossbar memristor is optimized to provide efficient read and write operations so that its performance is comparable to DRAM [30]. The area is optimized by using 3D crossbar memristive devices, which are implemented on top of DRAM. The design uses vertical select lines as bit lines for sensing. Two adjacent cells are used to represent a single CAM cell, where the match line (ML) is the sense line and the vertical select lines distribute input signal and its complementary among all rows. Since a single memristor device requires 4F2/n, the proposed design has a cell density of 8F2/n, where F is feature size and n is the number of vertical layers. The MPIM memory is designed to store non-write intensive data, e.g., reference data points for machine learning algorithms, learned weights for neural network and target graph structures to be traversed, thus minimizing the endurance issue of NVM technologies (e.g., up to 10^{12} write operations reported for memresitor devices [31]). We next explain how MPIM executes each in-memory operation.



Read/write/search operations

	Data	ML	SL/BL	SL'
Write	0	VDD	0	Vdd
	1	0	Vdd	0
Search	0	Pre-Vdd	0	Vdd
	1	Pre-Vdd	Vdd	0
Read		Pre-Vdd	-	Vdd

Figure 2. MPIM integration into a 3D structure

A. Nearest Neighbor Search

An important functionality that MPIM supports is k nearest neighbor (k-NN) search which has a broad range of application domains especially for Big Data applications [30]. k-NN finds k similar data points whose distances are minimum in its difference criteria over all data points. Although this algorithm is normally accelerated with GPGPUs [32-34], performance is dominated by the cost of data movement. For example, to search through 1 billion data points, it takes 150 GFLOP and 500G number of data movements [35]. Instead our design significantly reduces the data movement by utilizing processing-in-memory.

B.1. MPIM in Search Mode

Before executing the search operation, the searched data needs to be stored in MPIM banks. We call the stored data as *reference* values. Then, for a given data value called a *query*, MPIM searches for the most similar data points through all banks in parallel based on the hamming distance criteria as the difference metric [30]. As explained in Section 3.1, each memory bank has ***C*** crossbar memories. In order to enable the search functionality, crossbar memory is configured as a CAM. Figure 3 shows a detailed illustration of the CAM. A CAM block consists of ***N*** blocks, each block has ***m***-bit data, while each row corresponds to the match line (ML). In order to count the hamming distance for each row without large peripheral digital circuitry, MPIM exploits an analog sense amplifier along with a parallel counter and a minimum detector circuit [36].

After the buffer of each block is activated with the query value, the search operation is performed serially starting from the block located in least significant section, e.g. Block 1 in the figure. First ML precharges to Vdd. The input buffer of the first block activates in T_1 clock cycles and distributes the input data among all rows through vertical lines simultaneously. Then, a *sense amplifier* determines the number of mismatches in terms of the hamming distance, for the partial block of each row. In the next cycle, the *Controller* activates the second block, e.g. Block 2, in T_2 cycles by activating the input buffer. At the same time, a *parallel counter* accumulates the number of mismatches of the previous cycle, T_1, for each row. The search operations continue until all blocks are covered. Finally, a priority checker block selects the ***k*** rows which have the smallest hamming distance. Each bank of ***C*** CAMs returns ***C*k*** NN values. Thus, all the computed results of all ***B*** banks, i.e., ***C*B*k*** NN values, are merged in the dedicated registers, which can be accessed by the processor.

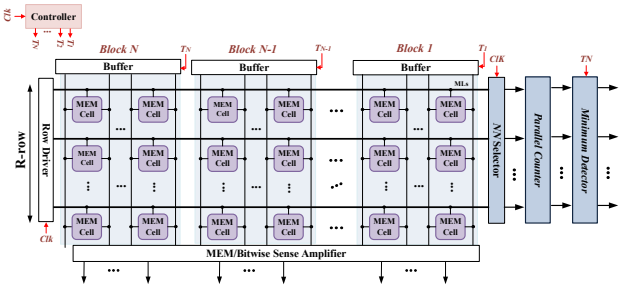


Figure 3. MPIM circuit for nearest neighbor search operation.

High performance search capability and the design scalability are the primary advantages of our technique. In contrast to the existing technique which searches for nearest neighbor data in linear or logarithmic time [5], our design can perform the computation using block serial, row parallel approach in constant time. In addition, the MPIM only needs a single sense amplifier, counter and comparator for a CAM to handle all blocks, thus significantly reducing the area overhead compared to all digital implementation. In the next subsection, we explain the design of the sense amplifier in more details.

B.2. Sense amplifier

In order to compute the hamming distance of a block in an efficient way, we leverage analog timing characteristics of the memristor device. In our design, a CAM consists of multiple memory cells where each row is connected to a ML. Any mismatch on the CAM cells start discharging ML. As the number of mismatches increases, the discharging speed increases. For example, a CAM line with 2-bit mismatch discharges significantly faster than CAM with 1-bit hamming distance. Thus, the sense amplifier can detect the number of mismatches, as estimated by the hamming distance, by sampling the ML discharging current every cycle.

However, there is no linear dependency between the ML discharging speeds and the number of hamming distance bits. In fact, the discharging speeds saturates as the number of mismatches increases. For example, in 32-bit CAM, 5 and 6 bits hamming distance have pretty similar ML discharging time, while there is a large time difference between having 1 and 2 bits hamming distance. This current saturation does not allow us to identify the hamming difference for the full bit line size. At the circuit level, the primary reason of this saturation is limited ML charge. In order to solve the saturation issue, the proposed design exploits two techniques. First, we split bit lines into the multiple blocks, i.e., Block *1* to *N*.

Figure 4 shows the sense amplifier architecture which can identify difference between 1, 2, 3 and 4 bits hamming distances. Our design periodically samples the ML

discharging current multiple times to identify the number of mismatches. In case of any mismatch in the CAM row, the ML voltage starts discharging and turns the M_1 transistor on. The M_1 current is mirrored to the M_2 branch. The sampled current is sent to DMUX which generates 4-bit digital signal representing the hamming distance (HD). A simple 2-bit wrap counter is used to keep track of clocks and control the DMUX output.

To guarantee the correct functionality of the proposed design in corner cases, we design the CAM and the sense circuitry by considering 10% of process variations on the transistors, i.e., the size and threshold voltage, and memristor resistance values. As explained in Section 3.2.1, the counted hamming distance is sent to the parallel counter to get a total number of mismatches for the full bit line size on each row. This parallel counter is designed to work with the output signals of the sense amplifier since the DMUX output uses its specified binary representation.

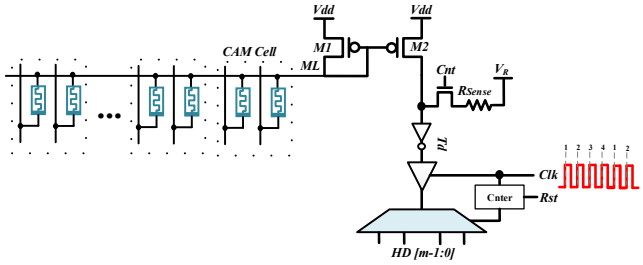


Figure 4. Sense amplifier for nearest neighbor search in MPIM

B. Bitwise Computation

The second key functionality supported by MPIM are bitwise operations. MPIM has the capability of performing *OR, AND*, and *XOR* operations inside the memory without any processing cores. In bitwise computation mode, MPIM can support either the *OR/AND* operations for multiple memory rows or the *XOR* operation for two rows in the same crossbar memory. Since the proposed technique can perform bitwise computation as fast as reading data from memory without any data movement and processing time, the program performance and energy efficiency can be improved significantly, especially for large input datasets.

C.1. MPIM in Bitwise Computation Mode & as Memory

In order to compute the bitwise operations, additional sense amplifiers, called bitwise sense amplifiers, are implemented at the tail of the bit lines of each crossbar memory. Figure 5 shows the architectural overview of the bitwise sense amplifier. When the bitwise operation processes two vector operands, the row driver activates the two corresponded rows (MLs). Then, a sense amplifier corresponding to each bit performs the computation based on the discharging current of the bit line (BL). The current can determine each bit value of a vector operand. If both memristor devices store a logic '1' which presents as having high resistance, the BL discharging current is close to zero, say $\mathbf{I_l}$. When both bits are '0' due to their low resistance, a large current, say $\mathbf{I_h}$, is detected. If only one of the two bits is '0', i.e., one of low resistance and the other of high resistance, the BL presents a middle range of current, $\mathbf{I_m}$ ($\mathbf{I_l < I_m < I_h}$). Thus, we can identify the bitwise operation result for a bit based on a voltage-based sense amplifier circuitry which leverages these current differences.

The current detection in a sense amplifier is performed using three different sense resistances, R_{AND}, R_{OR}, R_{MEM}. The resistance, R_{AND}, with large sense resistance detect the discharging current of $\mathbf{I_h}$. Similarly, another sense resistance, R_{OR}, can perform the OR operation by identifying when the discharging current is larger than $\mathbf{I_m}$. To support normal memory read operations, R_{MEM} with low sense resistance is used to detect any discharging current when a row is activated by the driver. The *XOR* operation does not require additional sense resistance. Instead, this functionality is implemented based on the results of *AND* and *OR*, i.e., M6 and M7 transistors as the following:

XOR= (>1-bit one) AND (<2-bit one).

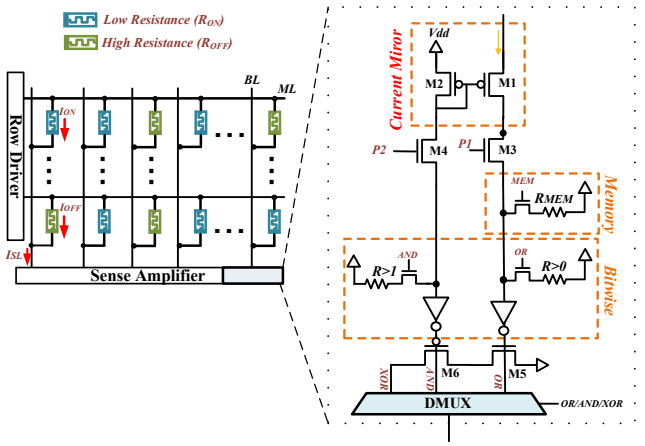


Figure 5. Sense circuitry for memory and bitwise computation.

Table 2. MPIM control signals to support memory and bitwise computation.

Computing/MEM Modes		P1	P2	OR	MEM	BL	ML
Bitwise Computation	*AND*	0	1	0	0	Connect to SA	Active on selected lines
	OR	0	1	1	0		
	XOR	1	1	1	0		
MEM		1	0	0	1	Connect to SA	Active on selected line

Table 2 shows the control signals and the ML/BL voltage for different MPIM operational modes. *P1*, *P2* and MEM control signals determine the MPIM mode by activating the sense amplifier corresponding to the *AND, OR* and memory operations, respectively. For the *XOR* operation, MPIM needs to activate both *P1* and *P2* signals. These signals also control the selector of the DMUX block to obtain the target result of the operation. Note that the delay of the DMUX block is negligible since the selector bit is activated much sooner than the actual computation on the sense resistances and transistors.

C.2. Multi-Row Operation Support

In order to more efficiently handle large datasets, our MPIM design also support multi-row bitwise operations for *AND* & *OR* by activating multiple memory rows. The processor can set multiple vector operands of an in-memory bitwise operation to the memory controller by either selecting the associated rows separately or assigning a bitmap which includes the target rows. For OR operation, we can enable the multi-row operations using the sense amplifier in the same way as in the two-operand case. For example, the row driver activates the MLs of the selected rows, then the cells with "1" value starts leaking to BL due to low resistance. The discharging currents are collected by the tail sense circuitry, so we can obtain "1" output of the OR operation whenever any resistance cell is low among all bits of the selected multiple rows. However, if many rows are activated, the result can be inaccurate due to the leakage current of the high resistance cells. For example, even though all stored values are '0', the leakage of all the high resistance cells to a bit line can be as high as the ON current threshold. Thus, the number of rows which can be supported by an *OR* operation is determined by the ratio of ON/OFF current of the memory cells. When considering 10% process variations on transistor size and resistance values in 5000 Monte Carlo simulation, we observed that the number of input vector operands cannot exceed 256 rows to ensure the correct *OR* computation.

We do not exploit the same strategy for the *AND* operation, since it needs to identify the case when all selected vectors have low resistance. Thus, instead of using the threshold voltage, we use the timing characteristic of the BL leakage current in a similar way used in the design of the hamming distance detection technique discussed in Section 3.2. For example, if at least one of the cells in a same BL stores '0', the BL current discharges faster than the case that all bits are '1'. In order to exploit the timing characteristics, we design sense circuit as shown in Figure 6. It samples the BL current based on the voltage value of the charged capacitor (V_K). The sampling time is set as the last time when all bits of activated rows are '1'. However, we cannot obtain clear distinction in timing for a large number of activated rows, since the timing difference saturates as the number of activated rows increase. Thus, our design allows the multi-row *AND* operation for up to 10 operands which samples in 2.4ns.

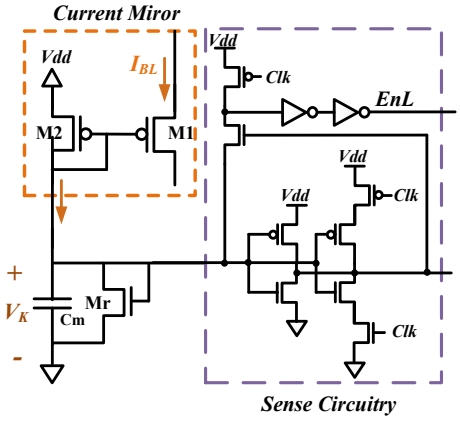


Figure 6. The sense circuitry for multi-row *AND* operation.

IV. Results

A. Experimental Setup

We compare the proposed MPIM architecture with implementations of applications running on state of the art processors AMD Radeon R9 390 GPU with 8 GB memory. In order to avoid the disk communications in the comparison, all the data used in the experiments is preloaded into 64GB, 2.1GHz DDR4 DIMMs. Power consumption is measured by Hioki 3334 power meter. We estimated performance, energy and area overheads of MPIM with Synopsys design compiler, and for circuit level simulation we use HSPICE with 45-nm technology.

To compare search operations, we implement k-NN algorithm which searches the nearest data point in terms of Euclidean distance using *OpenCL* [34] and run it on AMD Radeon R9 390. The GPU-based implementation improves the search performance by parallelizing multiple query requests and data searches. We use physical activity monitoring data set, named PAMPA2 [37], which includes 64-bytes data elements with measured values of 3D accelerometers positioned on arm, chest and ankle for 8 users. Each data element is labeled with the actual activity, e.g., sitting and walking. We cross-validated the accuracy of searched activities for each user by comparing the actual label of each data point to the label of the searched data in MPIM. On average 99.48% of hamming distance-based search operations of MPIM find the same data labels as GPU-based k-NN algorithm over all users.

To evaluate bitwise computation, we use micro benchmark which performs vector processing procedures based on SIMD. We use randomly generated dataset and the data access pattern does affect the performance. Thus, we tested MPIM efficiency with sequential and random access cases. In the sequential access case, since the CPU can exploit data locality, it has higher performance and energy efficiency. In contrast, the random access case shows the impact of the limited cache and memory bandwidth on conventional processor-memory architectures.

B. Dataset Size

The number of banks, ***B***, and the number of CAMs in each bank, ***C*** are configurable. These are main factors which affect MPIM overhead in terms of energy and performance. The overhead is due to the large buffer size that it requires in CAM mode to distribute the data to all rows. In our design, we use 1024 banks and 256 CAMs where each CAM stores 5KBytes, so that the simulated MPIM can store and load 1GBytes of data which can cover the largest size of PAMPA2 dataset.

Figure 8 shows the energy consumption and performance comparison of search operations running on two platforms, the GPU and the proposed MPIM. The algorithm searches the nearest data points for 16K queries for various data sizes from 16MB to 1GB. Our evaluation shows that the MPIM achieves 5.5**x** energy saving and 19**x** performance improvement as compared to the GPU-based k-NN approach. For large datasets size >1GB, the energy and performance of

the proposed MPIM is expected to further improve. The results show that data size affects. The proposed MPIM does not significantly degrade the energy and performance over the input dataset increase since this reduces the overhead of the data movement.

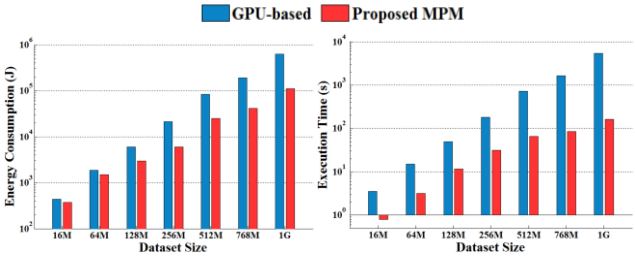


Figure 7. Energy consumption and performance of MPIM and GPU-based k-NN using different dataset sizes.

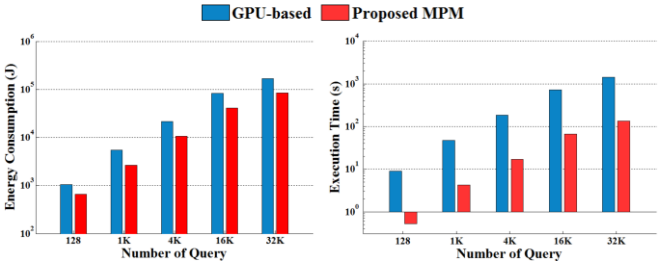
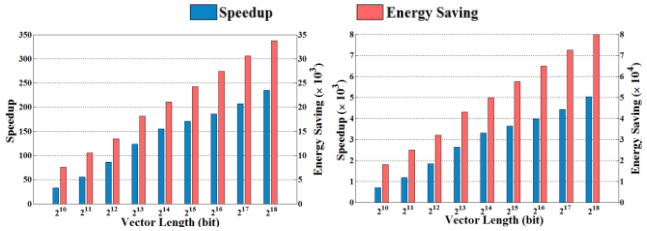


Figure 8. Energy consumption and performance of MPIM and GPU-based k-NN for multiple query processing.

Figure 9 shows the energy and performance comparison of the two approaches while varying the number of queries from 128 to 32K for the 512MB dataset. Even though the GPU-based computation can parallelize multiple queries as well, the performance and energy efficiency still decreases significantly due to memory overhead. In contrast, based on high performance of the search operation for the single query, our design can perform better even in the sequential searches of multiple queries.

C. *MPIM bitwise computation*

We next study the vector processing applications to show benefits of bit-level processing in MPIM. Since the bit size of a vector, *vector length*, is one factor which affects efficiency, we first show results for various lengths. Figure 10 shows the improvement in energy and speedup of MPIM computation over the GPU-based SIMD architecture. The x-axis shows the length of vector. The result shows that the proposed MPIM outperforms the GPU-based computation for both *AND* & OR operations. In addition, for larger vector length cases, the improvements are higher, since the CPU computation needs to compute the vectors of the large length sequentially by dividing the vector elements. In fact, the improvement starts saturating starting with 2^{14}-bit vector length, since the vector length is larger than the memory line size. For the vectors over the length of the memory word-line, MPIM splits the vectors and process them serially. This fact can be observed on Figure 10, where slop pf MPIM speed up reduces in large vectors. However, we observed that the proposed design can still process the data in a performance and energy efficient way.



(a) AND Operation (b) OR Operation

Figure 9. Speedup and energy saving of MPIM in OR/AND vector processing (2^{16} vectors) compared to SIMD

One major advantage of our MIPM design is that we can handle bitwise computations of multiple rows. As discussed in 3.3.2, the number of vectors that the MPIM can support depends on the operator type. For example, the MPIM supports the multi-row OR operation for up to 256 rows, while the *AND* operation is limited to 10 rows. you should mention this in intro paragraph to results section Table 3 compares the speed up and energy savings of MPIM with state-of-the-art PIM techniques in DRAM (DRAM-PIM [16]) and NVM (Pinatubo [17]) using 2^{16} vectors of 2^{14}-bit length. We compare the results for two different cases, where the data processing happens in sequential or random access cases. The results show that MPIM achieves significantly higher efficiency as compared to Pinatubo and DRAM-PIM. For example, in the AND operation comparison, the proposed design can improve speedup and energy consumption by 7.1**x** and 1.9**x** compared to Pinatubo [17], since we can also support the multi-row computations which the other technique does not support. In addition, we increase the supported number of rows for OR operations by 256 rows, which is 2**x** higher than the Pinatubo. As a result, the MPIM shows the improvement on speedup and energy consumption of 1.2**x** and 1.6**x**.

Table 3. Speedup and energy savings of MPIM as compared to Pinatubo [17] and DRAM-PIM [16]

		AND		*OR*	
# vectors		*Sequential*	*Random*	*Sequential*	*Random*
MPIM	*Energy*	$1.1*10^4$	$2.1*10^4$	$3.8*10^4$	$5.0*10^4$
	Speedup	121.6	154.4	$2.4*10^3$	$3.3*10^3$
Pinatubo	*Energy*	$0.6*10^3$	$1.1*10^3$	$2.9*10^4$	$4.1*10^4$
	Speedup	18.1	21.3	$1.3*10^3$	$2.0*10^3$
DRAM-PIM	*Energy*	$0.2*10^3$	$0.5*10^3$	$0.2*10^3$	$0.5*10^3$
	Speedup	4.9	8.3	4.9	8.3

D. *Area Overhead*

We evaluate the area overhead of the proposed MPIM with Synopsys Design Compiler to estimate the area of the peripheral circuitries, and NVsim simulator [38] to estimate the area of NVM memory. As shown in Figure 11, the area evaluation shows that the area overhead of MPIM is less than 5.1% over the entire area of MPIM which includes conventional NVM memory. MPIM peripheral circuitry requires 4.5% of the area for the search-based peripheral circuitry (comparator and counter) and 0.6% for the bitwise sense circuitry.

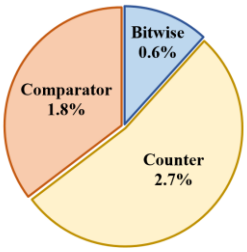


Figure 10. Area overhead breakdown of MPIM.

V. Conclusion

In this paper we propose multi-purpose in-memory processing design, MPIM. Our design uses crossbar resistive memory implemented in a 3D structure on top of conventional DRAM. MPIM handles several key operations including k nearest neighbor search and bitwise computation, in addition to acting as a normal NVM memory. Our experimental evaluation shows that the proposed MPIM improves the energy efficiency of the search operation by up to 5.5**x** and has a 19x of speed up as compared to the GPU-based measurements. When running bitwise operations, our design can achieve by at least 11000**x** energy saving and 62**x** speedup as compared to the state-of-art SIMD. We also show that our design also outperforms existing state-of-the-art in-memory processing techniques by 1.8**x** and 6.7**x** in terms of the energy and performance respectively. The total area overhead of MPIM is minimal.

Acknowledgements

This work was supported by NSF grant #1527034 and Jacobs School of Engineering UCSD Powell Fellowship.

References

[1] J. Gubbi, et al., "Internet of Things (IoT): A vision, architectural elements, and future directions," Future Generation Computer Systems, vol. 29, pp. 1645-1660, 2013.

[2] B. D. Rouhani, et al., "Automated Analysis of Streaming Big and Dense Data on Reconfigurable Platforms," ACM Transactions on Reconfigurable Technology and Systems (TRETS), 2016.

[3] B. D. Rouhani, et al., "SSketch: An automated framework for streaming sketch-based analysis of big data on fpga," in IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM), pp. 187-194, 2015.

[4] K. Hwang, et al., Distributed and cloud computing: from parallel processing to the internet of things: Morgan Kaufmann, 2013.

[5] C. C. del Mundo, et al., "NCAM: Near-Data Processing for Nearest Neighbor Search," in Proceedings of the International Symposium on Memory Systems, pp. 274-275, 2015.

[6] S. Beamer, et al., "Direction-optimizing breadth-first search," Scientific Programming, vol. 21, pp. 137-148, 2013.

[7] M. Imani, et al., "State-Feedback Control of Partially-Observed Boolean Dynamical Systems Using RNA-Seq Time Series Data." American Control Conference, 2016.

[8] M. Imani, et al., "Optimal state estimation for boolean dynamical systems using a boolean Kalman smoother," in IEEE Global Conference on Signal and Information Processing (GlobalSIP), pp. 972-976, 2015.

[9] R. Balasubramonian, et al., "Near-data processing: Insights from a MICRO-46 Workshop," Micro, IEEE, vol. 34, pp. 36-42, 2014.

[10] G. Loh, et al., "A processing in memory taxonomy and a case for studying fixed-function pim," in Workshop on Near-Data Processing (WoNDP), 2013.

[11] Q. Guo, et al., "A resistive TCAM accelerator for data-intensive computing," in Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture, pp. 339-350, 2011.

[12] M. Imani, et al., "Approximate Computing using Multiple-Access Single-Charge Associative Memory," IEEE Transactions on Emerging Topics in Computing (TETC), 2016.

[13] R. J. Gerrig, "The scope of memory-based processing," Discourse Processes, vol. 39, pp. 225-242, 2005.

[14] R. Nair, et al., "Active Memory Cube: A processing-in-memory architecture for exascale systems," IBM Journal of Research and Development, vol. 59, pp. 17: 1-17: 14, 2015.

[15] Y. Wang, et al., "ProPRAM: exploiting the transparent logic resources in non-volatile memory for near data computing," in Proceedings of the 52nd Annual Design Automation Conference, 2015, p. 47.

[16] V. Seshadri, et al., "Fast Bulk Bitwise AND and OR in DRAM," 2015.

[17] S. Li, et al., "Pinatubo: a processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories," in Proceedings of the 53rd Annual Design Automation Conference, p. 173, 2016.

[18] S. H. Pugsley, et al., "NDC: Analyzing the impact of 3D-stacked memory+ logic devices on MapReduce workloads," in Performance Analysis of Systems and Software (ISPASS), 2014 IEEE International Symposium on, 2014, pp. 190-200.

[19] Q. Guo, et al., "Ac-dimm: associative computing with stt-mram," in ACM SIGARCH Computer Architecture News, 2013, pp. 189-200.

[20] J. Ahn, et al., "PIM-enabled instructions: a low-overhead, locality-aware processing-in-memory architecture," in Proceedings of the 42nd Annual International Symposium on Computer Architecture, 2015, pp. 336-348.

[21] M. Imani, et al., "MASC: Ultra-low energy multiple-access single-charge TCAM for approximate computing," in 2016 Design, Automation & Test in Europe Conference & Exhibition (DATE), 2016, pp. 373-378.

[22] M. Imani, et al., "A low-power hybrid magnetic cache architecture exploiting narrow-width values," in 5th Non-Volatile Memory Systems and Applications Symposium (NVMSA), pp. 1-6, 2016.

[23] N. Khoshavi, et al., "Bit-Upset Vulnerability Factor for eDRAM Last Level Cache Immunity Analysis," in 17th International Symposium on Quality Electronic Design (ISQED), pp. 6-11, 2016.

[24] M. Imani, et al., "Low power data-aware STT-RAM based hybrid cache architecture," in 17th International Symposium on Quality Electronic Design (ISQED), 2016, pp. 88-94, 2016.

[25] M. V. Beigi, G. Memik, "TESLA: Using Microfluidics to Thermally Stabilize 3D Stacked STT-RAM Caches," IEEE International Conference on Computer Design, 2016.

[26] M. V. Beigi. G. Memik, "TAPAS: Temperature-aware Adaptive Placement for 3D Stacked Hybrid Caches," international Symposium on Memory Systems, 2016.

[27] Y. Kim, et al., "CAUSE: critical application usage-aware memory system using non-volatile memory for mobile devices," Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 690-696, 2015.

[28] M. Imani, et al., "Resistive Configurable Associative Memory for Approximate Computing." in Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 373-378, 2016.

[29] C. Xu, et al., "Overcoming the challenges of crossbar resistive memory architectures," in International IEEE Symposium on High Performance Computer Architecture (HPCA), pp. 476-488, 2015.

[30] Y. Gong, et al., "Iterative quantization: A procrustean approach to learning binary codes," in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 817-824, 2011.

[31] M.-J. Lee, et al., "A fast, high-endurance and scalable non-volatile memory device made from asymmetric Ta2O5− x/TaO2− x bilayer structures," Nature materials, vol. 10, pp. 625-630, 2011.

[32] H. Jégou, et al., "Searching with quantization: approximate nearest neighbor search using short codes and distance estimators," 2009.

[33] D. Qiu, et al., "GPU-accelerated nearest neighbor search for 3D registration," in Computer Vision Systems, ed: Springer, pp. 194-203, 2009.

[34] V. Garcia, et al., "K-nearest neighbor search: Fast GPU-based implementations and application to high-dimensional feature matching," in 17th IEEE International Conference on Image Processing (ICIP), pp. 3757-3760, 2010.

[35] Y. Deng, et al., "Content-based search of video using color, texture, and motion," in International Conference on Image Processing, pp. 534-537, 1997.

[36] L. G. Amaru, et al., "High speed architectures for finding the first two maximum/minimum values," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 20, pp. 2342-2346, 2012.

[37] A. Reiss, et al., "Creating and benchmarking a new dataset for physical activity monitoring," in Proceedings of the 5th International Conference on PErvasive Technologies Related to Assistive Environments, p. 40, 2012.

[38] X. Dong, et al., "Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, vol. 31, pp. 994-1007, 2012.