In-Memory Processing to Support Search-Based and Bitwise Computation

Mohsen Imani, Yeseong Kim, and Tajana Rosing
E-mail:{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 three important in-memory processing capabilities: i) addition/multiplication, ii) data searching for the nearest neighbor iii) bitwise operations including OR, AND and XOR by designing new analog sense amplifiers. The experimental results show that over application with addition/multiplication MPIM can achieve 9.2× energy efficiency improvement and 50.3× speedup as compared to a recent AMD GPU architecture. Similarly, MPIM can provide up to 5.5× energy savings and 19× speedup for the search operations. For bitwise vector processing, we present 11000× energy improvements with 62× speedup over the SIMD-based computation, while outperforming other state-of-the-art in-memory processing techniques.**

***Index Terms*—Processing in-memory, Non-volatile memory, Associative memories**

I. Introduction

In 2015, the number of smart devices around the world exceeded 25 billion. This number is expected to double by 2020 [**?**]. The rate of data generation by the Internet of things (IoT) will quickly overtake the capabilities of current computing systems. The need for systems that can efficiently handle such large volumes of streaming data is undeniable. This computational reduction is also essential in real time processes of varieties of fields including robotics [**?**], biology [1], etc. Running machine learning algorithms or multimedia applications on the general purpose processors, e.g. GPU, results in large energy and performance inefficiency. Many of these applications do not need highly accurate computation. Instead of doing all computation precisely, we can instead get the energy and performance advantages while accepting a slight loss in accuracy of computation [**?**], [**?**].
Computing in-memory can efficiently perform the computation without using processors. Associative memory, in a form of a lookup table, stores commonly seen patterns and retrieves them at runtime [**?**], [2], [**?**], [**?**], [**?**], [3], [4], [5], [6], [**?**], [7]. These memory blocks have application in a wide domain such as query processing [**?**], [**?**], text processing [**?**], search engine [**?**], [**?**], image processing [**?**], [**?**], [**?**], pattern recognition and classification [**?**], [**?**], [8].

In this work, we propose a new resistive memory design, called MPIM, which supports multiple in-memory processing operations along with traditional memory functionalities, i.e., read and write. We design a cost-efficient PIM by utilizing analog characteristics of emerging non-volatile memory (NVM) technology. Our design supports three major functionalities which are important to process the large amount of data. First, we support a fast in-memory addition and multiplication. Second, we support search operation to find the data of interest. Third, MPIM also supports bitwise operations even for multiple operands stored in different memory rows. MPIM supports the addition operation by making few changes to the sense amplifier, and then extends it to the multiplication operation by using an adaptive interconnect. When MPIM is configured to optimize the search, it performs a row-parallel search based on the timing difference of discharging current over the number of mismatched bits. For the bitwise computation, we exploit an analog sense amplifier which can act as an AND, OR, and XOR gate.

We evaluate the proposed design in a circuit-level simulation and compare energy and performance of MPIM to recent processor architectures as well as state-of-the-art PIM designs [9], [10]. The experimental results show that over application with addition/multiplication MPIM can achieve 9.2× energy efficiency improvement and 50.3× speedup as compared to GPU architecture. The results also show that running k-nearest neighbor (k-NN) tasks based on the MPIM search operation can improve energy and performance by 5.5× and 19×, respectively, as compared to GPU-based k-NN running on AMD Southern Island GPU. MPIM can also achieve 11000× energy efficiency improvement and 62× speedup for bitwise operations compared to a GPU-based SIMD machine.

II. Multi-Purpose In-Memory Processing

In this paper, we propose a multi-purpose resistive memory, called MPIM. Table I summarizes three key functionalities of MPIM with their application domains: i) load/store operations as an additional memory next to DRAM, ii) addition and multiplication, iii) nearest neighbor search operation and iv) bitwise computation. First, as a memory, MPIM is excellent for read-intensive data which is often used for in-memory computation, while other frequently-updated data would be kept in DRAM. Second, MPIM supports addition and multiplication in-memory. Third, MPIM also supports a block-serial, row-parallel search operation that can be used as a building block for several applications which need to identify a target data

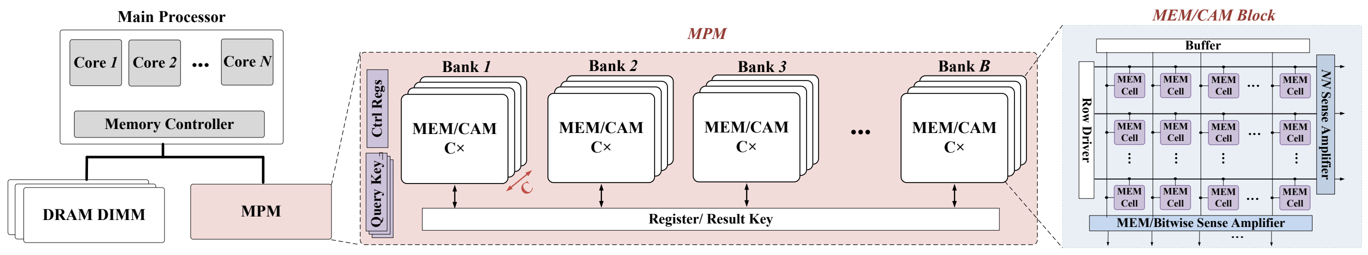

Fig. 1. The overview of MPIM architecture.

TABLE I
MPIM OPERATIONS AND APPLICATION EXAMPLES.

Operations	Applications
Addition/Multiplication	machine learning, security, multimedia, computer vision, Bioinformatics
Search	clustering algorithms, Statistical classifications, Database, Coding, Data compression, bioinformatic
Bitwise computation	Stream processing, Multimedia, Graph processing, communication/security

among all memory rows. Finally, MPIM supports bulk bitwise operations including OR, AND, and XOR. This capability allows us to use MPIM to efficiently process many streaming applications, which usually involve a large amount of bitwise computation, such as multimedia and graph processing.

Figure 1 shows an architectural overview 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 data in the same way to a conventional memory block. Stored data can be used for further processing. For example, the main processor can make a request for fetching the data directly 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 a CAM mode for search.

III. Experimental 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 test the efficiency of MPIM over addition/multiplication, we compare the efficiency of and GPU by running four general OpenCL applications including: *Sobel, Robert, Fast Fourier transform (FFT)* and *DwHaar1D*. For image processing we use random images from *Caltech 101* [11] library, while for non-image processing applications inputs are generated randomly. Majority of these applications consists of additions and multiplications. The other common operations such as square root has been approximated by these two functions in OpenCL code.

B. Nearest Search

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 3 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× energy saving and 19× 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 3 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 4 shows the improvement in energy and speedup of MPIM computation over the GPU-based SIMD architecture. The x-axis shows the

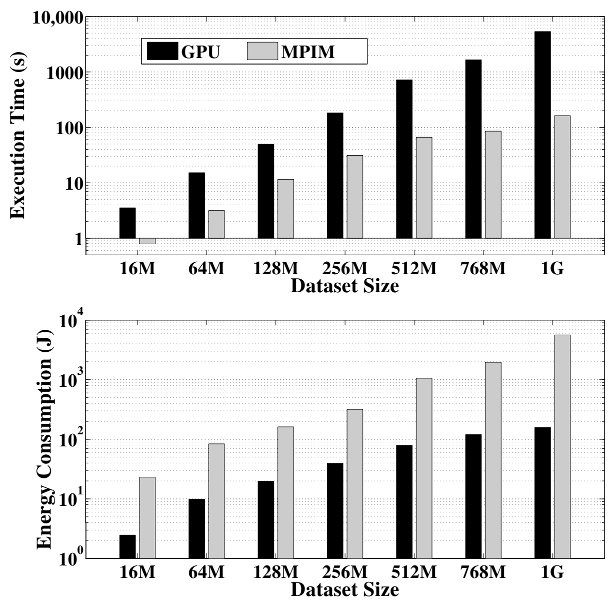

Fig. 2. Energy consumption and performance of MPIM and GPU-based k-NN using different dataset sizes.

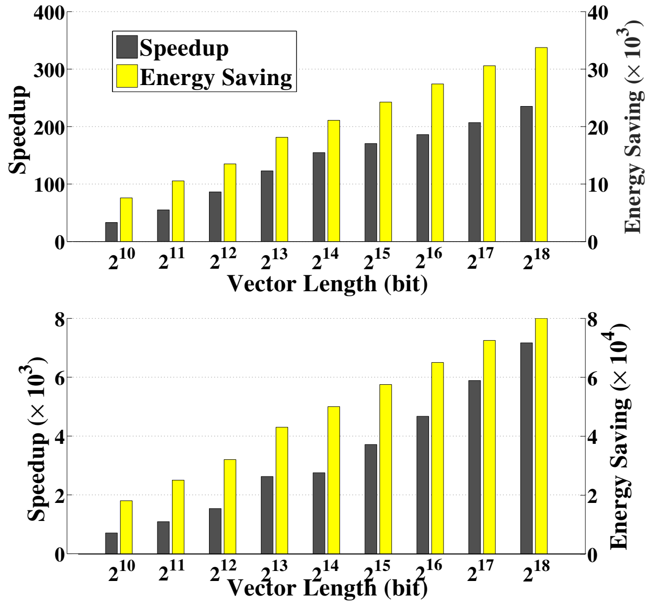

Fig. 4. Speedup and energy saving of MPIM in OR/AND vector processing (2^{16} vectors) compared to SIMD.

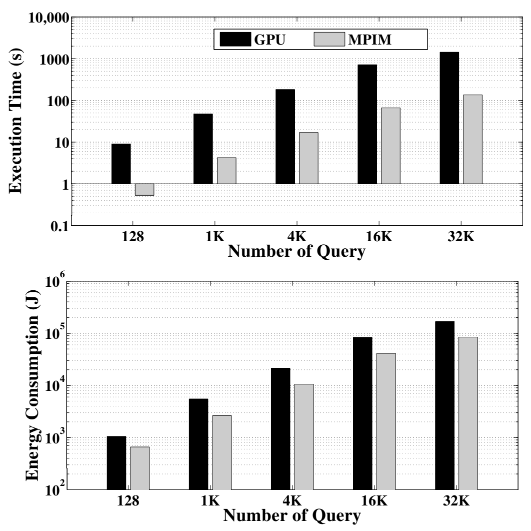

Fig. 3. Execution time and energy consumption of MPIM and GPU-based k-NN for multiple query processing.

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.

One major advantage of our MIPM design is that we can handle bitwise computations of multiple rows. As discussed in Section **??**, 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 II compares the speed up and energy savings of MPIM with state-of-the-art PIM techniques in DRAM (DRAM-PIM [9]) and NVM (Pinatubo [10]) using 2^{16} vectors of $2^1$4-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 $7.1\times$ and $1.9\times$ compared to Pinatubo [10], 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 $2\times$ higher than the Pinatubo. As a result, the MPIM shows the improvement on speedup and energy consumption of $1.2\times$ and $1.6\times$.

IV. Acknowledgment

This work was supported by NSF grants 1730158 and 1527034.

References

[1] M. Imani and U. Braga-Neto, "Optimal state estimation for boolean dynamical systems using a boolean kalman smoother," in *Signal and*

TABLE II
SPEEDUP AND ENERGY SAVINGS OF MPIM AS COMPARED TO PINATUBO [17] AND DRAM-PIM [16].

Operations		**AND**		**OR**	
Number of 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

Information Processing (GlobalSIP), 2015 IEEE Global Conference on, pp. 972–976, IEEE, 2015.

[2] M. Imani, A. Rahimi, and T. S. Rosing, "Resistive configurable associative memory for approximate computing," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2016*, pp. 1327–1332, IEEE, 2016.

[3] M. Imani, S. Patil, and T. Rosing, "Approximate computing using multiple-access single-charge associative memory," *IEEE Transactions on Emerging Topics in Computing*, 2016.

[4] M. Imani *et al.*, "Acam: Approximate computing based on adaptive associative memory with online learning," in *IEEE/ACM ISLPED*, pp. 162–167, 2016.

[5] M. Imani, S. Patil, and T. S. Rosing, "Masc: Ultra-low energy multiple-access single-charge tcam for approximate computing," in *Proceedings of the 2016 Conference on Design, Automation & Test in Europe*, pp. 373–378, EDA Consortium, 2016.

[6] M. Imani *et al.*, "Multi-stage tunable approximate search in resistive associative memory," *IEEE TMSCS*, 2017.

[7] M. Imani *et al.*, "Exploring hyperdimensional associative memory," in *IEEE HPCA*, IEEE, 2017.

[8] M. Imani *et al.*, "Mpim: Multi-purpose in-memory processing using configurable resistive memory," in *IEEE ASP-DAC*, pp. 757–763, IEEE, 2017.

[9] V. Seshadri, K. Hsieh, A. Boroum, D. Lee, M. A. Kozuch, O. Mutlu, P. B. Gibbons, and T. C. Mowry, "Fast bulk bitwise and and or in dram," *IEEE Computer Architecture Letters*, vol. 14, no. 2, pp. 127–131, 2015.

[10] S. Li, C. Xu, Q. Zou, J. Zhao, Y. Lu, and Y. Xie, "Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories," in *Design Automation Conference (DAC), 2016 53nd ACM/EDAC/IEEE*, pp. 1–6, IEEE, 2016.

[11] "Caltech Library." http://www.vision.caltech.edu/Image_Datasets/Caltech101/.