Hybrid Digital-Digital In-Memory Computing

Muhammad Rashedul Haq Rashed*, Sumit Kumar Jha†, Fan Yao*, and Rickard Ewetz*
*Department of Electrical and Computer Engineering, University of Central Florida, Orlando, USA
†Department of Computer Science, University of Texas at San Antonio, San Antonio, USA
rashed09@knights.ucf.edu, sumit.jha@utsa.edu, {fan.yao, rickard.ewetz}@ucf.edu

***Abstract*—In-memory computing (IMC) using emerging non-volatile memory promises exascale computing capabilities for a number of data-intensive workloads. The state-of-the-art solution to accelerating high assurance applications is based on digital in-memory computing. Digital in-memory computing can be WRITE-based or READ-based, i.e., logic is evaluated while switching or without switching the state of the non-volatile resistive devices. All prominent studies for accelerating matrix-vector multiplication (MVM) based applications utilize a single digital logic style. However, we observe that WRITE-based and READ-based digital in-memory computing are advantageous for dense and sparse matrices, respectively. In this paper, we propose a new computing paradigm called hybrid digital-digital in-memory computing paradigm. The paper also introduces automated synthesis tool for mapping computation to a hybrid architecture. The key idea is to first decompose the matrix into dense and sparse blocks. Next, bit-slicing is used to further decompose the dense blocks into sparse and dense parts. The dense (sparse) blocks are mapped to WRITE-based (READ-based) digital in-memory accelerators. The proposed paradigm is evaluated using 12 applications from various domains. Compared with WRITE-based IMC, the hybrid digital-digital paradigm improves energy and speed with 13X and 20X at the expense of increasing the area with 151X. Compared with READ-based IMC, the hybrid paradigms improves energy, speed, and area with 264X, 198X, and 2996X, respectively.**

I. Introduction

Exascale simulation is a pathway to continued scientific progress. However, scientific computing is facing energy-efficiency and data movement challenges [13]. With limited performance improvements expected from further technology scaling, emerging computing paradigms have recently attracted an immense amount of attention.

In-memory computing (IMC) using non-volatile memory is viewed as one of the most promising approaches to accelerate exascale computing applications in the near few years. Non-volatile memory are two terminal devices with programmable resistance, including resistive random access memory (ReRAM), phase change memory (PCM), spin-transfer torque magnetic random access memory (STT-MRAM) [12]. By integrating the memory devices into dense crossbar arrays, various mathematical kernels can be executed extremely energy-efficiently. Accelerating matrix-vector multiplication (MVM) operations are of particular interest as they are the dominating workload of most scientific computing applications. The simulation of physical systems is modeled using partial differential equations (PDEs). In each time step, a system of linear equations is required to be solved using Kyrlov subspace methods, which involve iteratively performing matrix-vector multiplication operations [5].

This work was in part supported by NSF awards CNS-1908471, CNS-2008339, CCF-1822976, CCF-2113307, DARPA cooperative agreement #HR00112020002 and ONR grant #N000142112332.

Noteworthy efforts have been devoted to accelerating MVM operations using analog and digital in-memory computing paradigms [3, 4, 8, 11]. While analog in-memory computing is extremely energy-efficient, it cannot deliver the deterministic precision required by scientific computing applications. A number of digital logic styles have been developed for accelerating MVM operations with high precision. The logic styles specify how Boolean logic is encoded using the state of the non-volatile devices and the analog input/output signals. Logic styles for digital in-memory computing are either READ-based or WRITE-based [3, 8]. WRITE-based logic styles involve switching the state of the non-volatile memory devices, whereas READ-based styles are capable of evaluating the logic without switching the state. While previous studies have mainly focused on determining the ideal logic style and architecture for accelerating MVM operations, we instead observe that READ-based and WRITE-based IMC are advantageous for different types of matrices. WRITE-based IMC performs very well for dense matrices, while READ-based IMC performs well for sparse matrices. Consequently, we speculate that it would be advantageous to combine READ-based and WRITE-based in-memory computing.

In this paper, we propose a new computing paradigm called hybrid digital-digital in-memory computing. We also present an automated synthesis tool capable of mapping computation to an in-memory computing platform. The synthesis tool first decomposes matrix-vector multiplication into dense and sparse parts using a *blocking* algorithm. Next, the dense parts are further decomposed into dense and sparse parts in the value domain using *bit-slicing*. The sparse and dense parts are accelerated using READ-based and WRITE-based computing, respectively. The experimental evaluation is performed using 12 applications from the SuiteSparse matrix collection [2]. Compared with READ-based computing, the hybrid digital-digital paradigm improves energy, speed, and area with 264X, 198X, and 2996X. Compared with WRITE-based computing, the hybrid digital-digital paradigm improves energy and speed with 13X and 20X at the expense of 151X overhead in area.

The remainder of the paper is organized as follows: preliminaries are provided in Section II. The hybrid digital-digital in-memory computing paradigm is outlined in Section III. Section IV discusses the synthesis tool. The experimental results are detailed in Section V. The paper is concluded in Section VI.

II. Preliminaries

In this section, we first explain the WRITE-based in-memory computing and the READ-based in-memory computing. Next, we exemplify a dense system and a sparse system and evaluate the systems using the two in-memory logic styles.

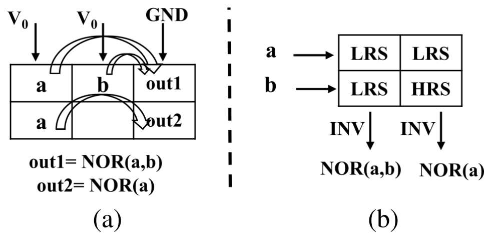

Fig. 1: Evaluation of Boolean logic using (a) WRITE-based MAGIC logic family and (b) READ-based OR-plane logic family. Each block represents a memristor inside the crossbar.

A. *WRITE-based digital in-memory computing*

Figure 1(a) illustrates a WRITE-based in-memory computing style – MAGIC [8]. MAGIC is an in-situ computation method, where the input and output operands are stored inside the memory. A MAGIC operation has two steps: an initialization step and an execution step. In the initialization step, the input operands are stored into the same memory row and the output operands are initialized to 1. In the execution step, controlled voltages are applied and Boolean logic gates are realised by switching of the state of the output memristors, i.e., using a special WRITE operation. Figure 1(a) shows the execution of a two-input NOR gate and an inverter.

B. *READ-based digital in-memory computing*

Figure 1(b) illustrates a READ-based in-memory computing style – the OR-plane logic [3]. The OR-plane logic realizes an arbitrary input OR gate by first setting the memristors in a bitline either to the low resistance state (LRS) or the high resistance state (HRS). Next, the input operands are applied as binary voltages to the wordlines of the crossbar to evaluate the intrinsic OR-gates. An inverter in the peripheral can realize an arbitrary input NOR gate. Hence, the NOR-gates are evaluated using READ operations.

C. *Case studies with dense and sparse systems*

We make the observation that different in-memory computing paradigms perform differently for dense and sparse matrices. The WRITE-based MAGIC computing is ideal for dense matrices where high order of parallelism is offered. State-of-the-art MAGIC synthesis tools achieves high throughput by performing single instruction, multiple data (SIMD) operations in parallel [1]. On the other hand, the size of the computational kernel and the routing cost both scale up very rapidly for the READ-based in-memory computing while evaluating a dense system. However, the WRITE-based computing cannot avoid redundant multiplications with zeros within sparse system. Therefore, READ-base computing can achieve superior energy-latency performance by reducing the computational workload.

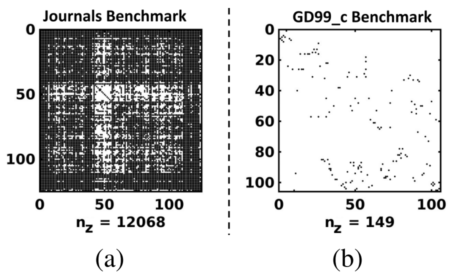

Fig. 2: Systems from SuiteSparse matrix collection [2]: (a) a dense system and (b) a sparse system.

Using the experimental setup explained in Section V, we perform a case study of associated energy and latency cost of in-memory computing for the systems in Fig. 2. Fig. 2(a) shows a dense system with 78.5% non-zero values. And, Fig. 2(b) illustrates a sparse system with only 1.4% non-zero values. The evaluation results are presented in Figure 3.

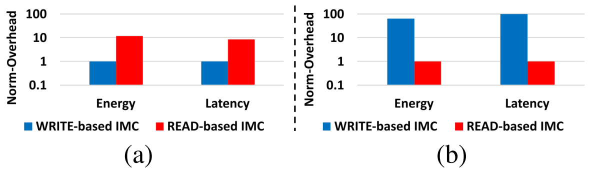

Fig. 3: Comparative energy and latency overhead for the systems in Fig. 2: (a) the *Journals* benchmark and (b) the *GD99_c* benchmark.

For the dense *Journals* system, the results show that the WRITE-based computing achieves 12X and 9X energy and latency improvements respectively over the READ-based computing. Conversely, for the sparse *GD99_c* system, the READ-based computing is 64X and 99X more energy and latency efficient respectively over the WRITE-based computing. Note that, the READ-based computing is always more expensive in terms of area due to its lower cell utilization and no cell-reuse.

The results indicate that neither of the two in-memory computing approaches is best for all the physical systems. Additionally, many physical systems have localized dense and sparse segments, as shown by an example system in Fig. 4(a). This motivates us to develop a hybrid digital-digital in-memory computing platform, where we evaluate the dense segments using WRITE based in-memory computing and the sparse segments using READ based in-memory computing.

III. Hybrid Digital-Digital In-Memory Computing

The overview of the hybrid computational decomposition is illustrated in Figure 4(b). We propose a two-level decomposition scheme for the hybrid paradigm.

Most physical systems, while being sparse overall, contain localized dense areas. We define these areas as blockable dense segments. In the first level of hybridization, we extract these dense segments through blocking, which is explained in Section IV-A. This leaves us with a very sparse original matrix, which is an automatic candidate for the READ-based in-memory computing. Next, the intuitive choice for the dense blocks is the WRITE-based in-memory computing. However,

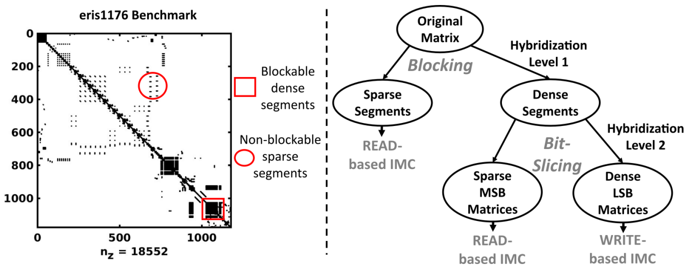

Fig. 4: (a) System with localized dense and sparse segments. (b) Hybrid digital-digital computational decomposition.

we make the observation that a dense matrix segment in a physical system can be further decomposed into a sparse most-significant bits (MSBs) matrix and a dense least-significant bits (LSBs) matrix. We achieve this by bit-slicing the operands of the original dense matrix [4]. Finally, we map the sparse MSB matrix into READ-based in-memory computing kernel and the dense LSB matrix into the WRITE-based in-memory computing kernel. In the following section, we develop the synthesis for the hybrid digital-digital in-memory computing.

IV. Synthesis

Our synthesis flow consists of a computational decomposition stage and a hardware binding stage. The computational decomposition stage can be further divided into two hybridization steps: one blocking step and one bit-slicing step.

A. *Hybridization level 1: blocking*

The aim of the blocking step is to extract the localized dense blocks from the original systems. The synthesis tool adopts a $\mathcal{O}(n_z)$ blocking algorithm similar to [4], where n_z is the number of non-zero operands in the original matrix.

The synthesis tool takes the system matrix M, operands bit-precision n, a fixed blocking size $(k \times l)$ and a block density threshold d as inputs. The density threshold parameter, d, is a design choice which dictates the total number of blocks and the density quality of the blocks extracted. To find a potential block, the synthesis tool traverses through the system. This traversal is very efficient for a sparse matrix because the tool navigates the original matrix only in its non-zero operand locations. When the synthesis tool selects a non-zero element of index (i, j), it explores a block space of $(k \times l)$ starting from the location of (i, j). If the selected block space meets the density threshold requirement d, the algorithm extracts the block from the original matrix and moves to the second layer of hybridization, which is explained in the next section. If the density threshold is not met, the algorithm moves onto the next non-zero operands. At the end of the block extraction, the remaining sparse matrix M is forwarded to the READ based in-memory computing kernel.

B. *Hybridization level 2: bit-slicing*

The goal of the bit-slicing step is to decompose a dense matrix into a sparse MSB matrix and a dense LSB matrix. Next, the matrices are mapped into the READ-based and WRITE-based in-memory computing kernels respectively.

We speculate that, different bit-slicing methods will yield different overall energy costs. Let us assume that the operands of the dense matrix are represented using n bits. When we extract p MSBs from the matrix operands, where $p \ll n$, we get a sparse MSB matrix and a dense LSB matrix. With ascending values of p, the MSB matrix converts from a very sparse matrix to an incrementally denser matrix. We show an example of different bit-slicings with the *Journals* benchmark in Figure 5. The Figure shows the sparsity patterns of the MSB matrix that is extracted from the "originally" dense matrix for variable values of p. Our aim is to select a pair of MSB and LSB matrices so as to minimize the overall energy cost.

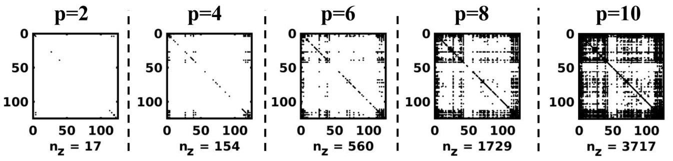

Fig. 5: Sparsity patterns of the MSB elements of *Journals* benchmark for variable bit-slicing width, p.

We perform a case study and evaluate the associated energy-latency cost of different bit-slicings for the *Journals* benchmark. We present the results of the case-study in Fig. 6. The results show that an energy minima occurs for bit-slicing the original matrix at $p = 7$. The synthesis tool aims to determine this minima point for each incoming dense blocks. Note that, the area overhead always increases with increment of p, as more area-expensive READ-based in-memory computing kernels are introduced.

To select a bit-slicing width p that yields the energy minima, the synthesis tool performs a binary search. The energy parameter, E, is initially set to infinity. Next, the function selects $p = n/2$ and performs a bit-slicing. The energy cost is estimated for the resultant MSB and LSB matrices and E is updated with this new value. Then, the binary search is continued in the direction, $\frac{\partial E}{\partial p} < 0$.

Upon termination, the synthesis tool returns an energy-optimized MSB matrix and an LSB matrix, which are then hardware bound into the READ and the WRITE-based in-memory computing kernels respectively.

C. *Hardware binding*

In this section, we describe the hardware binding method of the decomposed sparse and dense kernels of the hybrid

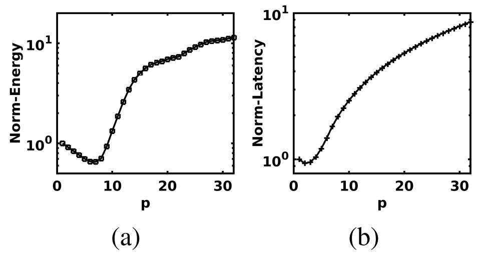

Fig. 6: Overhead of *Journals* benchmark for different bit-slicing widths, p: (a) normalized energy vs. p and, (b) normalized latency vs. p. All results are normalized with respect to purely WRITE-based computing. The energy minima occurs at bit-slicing width, $p = 7$.

TABLE I: Overview of SuiteSparse benchmarks [2].

Applications	Systems	Matrix Dimensions	#Non-zeros
bcsstk34	Structural Problem	588×588	21418
eris1176	Power Network Problem	1176×1176	18552
mycielskian12	Undirected Graph	3071×3071	407200
raefsky1	Computational Fluid Dynamics	3242×3242	293409
fxm3_6	Optimization Problem	5026×5026	94026
benzene	Theoretical/Quantum Chemistry	8219×8219	242669
bcsstk33	Structural Problem	8738×8738	591904
graham1	Computational Fluid Dynamics	9035×9035	335472
net25	Optimization Problem	9520×9520	401200
bundle1	Computer Graphics/Vision	10581×10581	770811
Si10H16	Theoretical/Quantum Chemistry	17077×17077	875923
pkustk06	Structural Problem	43164×43164	2571768

computing paradigm. The synthesis tool evaluates the decomposed dense blocks using the WRITE-based MAGIC in-memory computing. To bind the dense block computations into hardware, we utilize the state-of-the-art row-wise MAGIC mapping algorithm developed in [1]. For evaluating the sparse matrices using the READ-based in-memory computing, the synthesis tool utilizes the OR-plane logic based STREAM framework [10]. The STREAM framework optimizes the sparse MVM computation by converting the baseline netlist into high fan-in OR/NOR gates using the DAGON algorithm [7]. Next, the OR/NOR gates are mapped into OR-plane logic.

V. Experimental Evaluation

We evaluate the hybrid paradigm with a number of benchmarks from various domains of physical systems. We compare the performance of the proposed paradigm with purely READ-based in-memory computing and purely WRITE-based in-memory computing paradigms.

The overall hybrid architecture consists of several hybrid accelerators, each divided into a READ-based in-memory computing kernel and a WRITE-based in-memory computing kernel. The WRITE and READ kernels adopt the state-of-the-art row-parallel SIMPLER [1] framework and the staircase structured STREAM framework [10] respectively. The architectural area and power costs are appropriately adapted from [6, 9, 11].

For hybridization level 1, we set a block size of 128×128 in accordance with the crossbar size to ensure maximum parallelism opportunity for dense blocks. We set the density threshold $d = 10\%$. For fixed-point arithmetic, we set the precision $n = 32$ bits for the matrix-vector multiplication operands.

For experimental evaluation of physical systems, we select 12 benchmarks from the SuiteSparse matrix collection [2]. The overview of the benchmarks are presented in Table I. All of the systems are sparse, but contain localized dense segments.

We evaluate the energy and latency cost of the benchmarks using the WRITE-based MAGIC logic, READ-based OR-plane logic and the proposed hybrid digital-digital platform. The results of the evaluation are presented in Figure 7. The experimental results show that the hybrid paradigm achieves 13X and 20X improvements of energy and latency over the WRITE-based in-memory computing, at the expense of 151X overhead in area. Also, the hybrid paradigm achieves 264X, 198X and 2996X improvements of energy, latency and, area respectively over the READ-based in-memory computing. The experimental results confirm that the proposed hybrid paradigm is significantly more efficient than the state-of-the art READ and WRITE-based in memory computing platforms for evaluating the applications from physical systems.

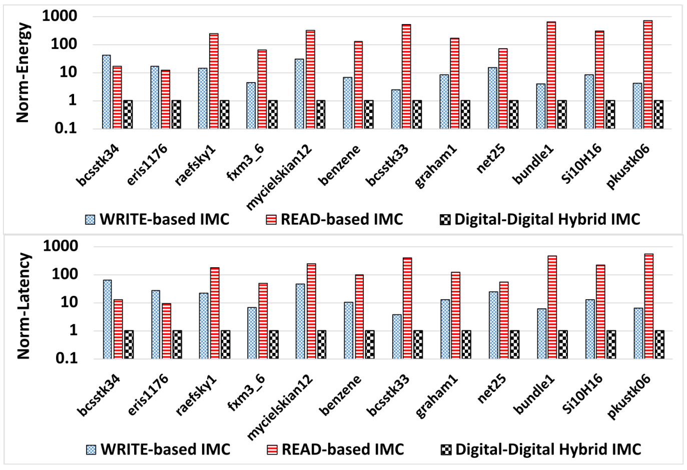

Fig. 7: Evaluation on SuiteSparse benchmarks.

VI. Summary and Future Work

In this paper, we present the observation that the READ-based in-memory computing is more efficient in evaluating sparse systems and the WRITE-based in-memory computing has superior performance for dense systems. We exploit this insight to propose a hybrid digital-digital in-memory computing platform that can evaluate physical systems more efficiently than both the state-of-the-art READ/WRITE based in-memory computing platforms. In future work, we aim to explore hybridization of different logic styles for new applications.

References

[1] R. Ben-Hur, R. Ronen, A. Haj-Ali, D. Bhattacharjee, A. Eliahu, N. Peled, and S. Kvatinsky. Simpler magic: Synthesis and mapping of in-memory logic executed in a single row to improve throughput. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(10):2434–2447, 2019.

[2] T. A. Davis and Y. Hu. The university of florida sparse matrix collection. *ACM Transactions on Mathematical Software (TOMS)*, 38(1):1–25, 2011.

[3] A. Dehon. Nanowire-based programmable architectures. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 1(2):109–162, 2005.

[4] B. Feinberg, U. K. R. Vengalam, N. Whitehair, S. Wang, and E. Ipek. Enabling scientific computing on memristive accelerators. In *2018 ACM/IEEE 45th ISCA*, pages 367–382. IEEE, 2018.

[5] M. R. Hestenes, E. Stiefel, et al. *Methods of conjugate gradients for solving linear systems*, volume 49. NBS Washington, DC, 1952.

[6] M. Imani, S. Gupta, Y. Kim, and T. Rosing. Floatpim: In-memory acceleration of deep neural network training with high precision. In *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*, pages 802–815. IEEE, 2019.

[7] K. Keutzer. Dagon: Technology binding and local optimization by dag matching. In *Proceedings of the 24th ACM/IEEE Design Automation Conference*, pages 341–347, 1987.

[8] S. Kvatinsky, D. Belousov, S. Liman, G. Satat, N. Wald, E. G. Friedman, A. Kolodny, and U. C. Weiser. Magic—memristor-aided logic. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 61(11):895–899, 2014.

[9] S. Kvatinsky, M. Ramadan, E. G. Friedman, and A. Kolodny. Vteam: A general model for voltage-controlled memristors. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 62(8):786–790, 2015.

[10] M. R. H. Rashed, S. Thijssen, S. K. Jha, F. Yao, and R. Ewetz. Stream: Towards read-based in-memory computing for streaming based data processing. In *Proceedings of the 27th Asia and South Pacific Design Automation Conference*, 2022.

[11] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar. Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. *ACM SIGARCH Computer Architecture News*, 44(3):14–26, 2016.

[12] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams. The missing memristor found. *nature*, 453(7191):80–83, 2008.

[13] W. A. Wulf and S. A. McKee. Hitting the memory wall: implications of the obvious. *ACM SIGARCH computer architecture news*, 23(1):20–24, 1995.