PIM-Quantifier: A Processing-in-Memory Platform for mRNA Quantification

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

Shaahin Angizi
School of Electrical, Computer and energy Engineering
Arizona State University
Tempe, USA
sangizi@asu.edu

Naima Ahmed Fahmi
Department of Computer Science
University of Central Florida
Orlando, USA
fnaima@knights.ucf.edu

Wei Zhang
Department of Computer Science
University of Central Florida
Orlando, USA
wzhang.cs@ucf.edu

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

2021 58th ACM/IEEE Design Automation Conference (DAC) | 978-1-6654-3274-0/21/$31.00 ©2021 IEEE | DOI: 10.1109/DAC18074.2021.9586144

***Abstract*—Processing-in-memory (PIM) architecture has been considered as a promising solution for the "memory-wall" issue in many data-intensive applications, especially in bioinformatics. Recent works of developing PIM for genome alignment and assembling have achieved tremendous improvement, while another important genome analysis - mRNA quantification has not been explored. Efficient and accurate mRNA quantification is a crucial step for molecular signature identification, disease outcome prediction and drug development. In this paper, for the first time, we propose a SOT-MRAM based PIM platform, named *PIM-Quantifier*, for efficient mRNA quantification. A PIM-friendly alignment-free quantification algorithm is first proposed. Then, we present the optimized PIM architecture/circuit designs and mapping method to efficiently accelerate mRNA quantification. Extensive experiments show that PIM-Quantifier significantly improves mRNA quantification performance than CPU and recent other PIM platforms in efficiency defined as throughput/power.**

***Index Terms*—Processing-in-memory, mRNA-seq, MRAM**

I. Introduction

According to the central dogma of molecular biology, a gene contains exons and introns in its structure, where coding exons are translated into protein. A single gene can encode a set of distinct proteins that participate in diverse biological functions by producing multiple transcripts (i.e., mRNA) with different combinations of exons. To better understand the biological functions and identify important molecular signatures for disease progression prediction and drug development, efficient and accurate transcript quantification with large-scale mRNA-sequencing (RNA-seq) data is crucially important [1], [2]. The high throughput RNA-seq technology is capable of measuring transcript expression by mapping tens of millions mRNA (or DNA) short reads (Fig.1(a)) to tens of thousands of annotated genes and each short read contains hundreds of *mRNA base pairs* (bps). The normalized read coverage on genes or transcripts represents their expression levels.

A typical transcript quantification with RNA-seq requires alignment of short reads to the whole genome or transcriptome before estimating the abundance, which is extremely time-consuming. For example, aligning 30 million short reads from one sample to the reference genome, using the widely used software program TopHat2 [3] takes 28 CPU hours, while quantification with the companion programs (e.g., Cufflinks [4]) takes another 1-2 CPU hours. Since a read can be mapped to multiple positions, ignoring the full base-to-base alignment of the reads can significantly increase alignment efficiency, and hence, the quantification too. An alignment-free technique [1] has been developed recently to solve the above issue. As shown in Fig.1(b), the technique only focuses on determining the transcripts from which the reads are generated, not the exact location. Without sacrificing the overall accuracy, this approach uses *k-mer* based counting algorithms where each transcript is split into *k-length* (bps) substrings to make it mappable with short reads efficiently and accurately. This striking idea introduces several novel bioinformatics tools (e.g., Kallisto [1] and Salmon [2]) which can quantify mRNA abundances (Fig.1(c)) without the exact position-wise alignment in a relatively short amount of time. However, there is an intrinsic need to map each short read to hundreds of thousands of transcripts for mRNA quantification, which still takes a large amount of computational resources. In this work, we aims to develop an efficient and fast hardware accelerator for such compute- and data-intensive alignment-free mRNA quantification process, which is a crucial step for molecular signature identification, disease progression prediction and drug development, etc.

In the meantime, Processing-in-Memory (PIM) architecture is being introduced widely in the past two decades to solve the memory wall bottleneck and improve processing time by exercising parallel computing [5]–[7]. Moreover, PIM has been demonstrated as a promising candidate to accelerate data-intensive applications, especially for neural-network and

This work is supported in part by the National Science Foundation under Grant No.2005209, No.2003749

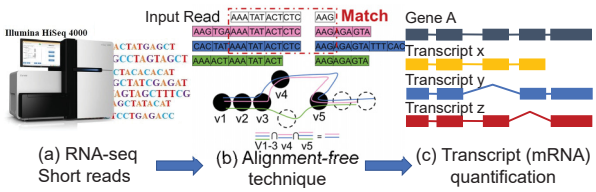


Fig. 1. fast alignment-free mRNA transcript quantification is a crucial step for molecular signature identification, disease progression prediction and drug development, etc.

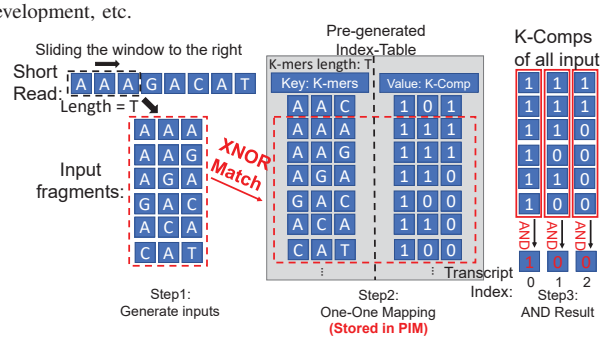


Fig. 2. mRNA quantification-in-memory: Index-table is on-time pre-computed and will be continuously used to process new incoming short reads. The key operations in the algorithm are XNOR based match and AND functions.

bioinformatics. Existing prior works only explored how to leverage PIM for DNA alignment and DNA assembling [8]–[11]. How to efficiently leverage PIM architecture to accelerate important mRNA quantification has yet been explored.

Our contributions in this work are summarized as following:
(1) To the best of our knowledge, we are the first to propose a PIM-friendly mRNA quantification algorithm, which converts the complex graph processing based algorithm into primary bulk bit-wise logic operations supported by most PIM architectures.
(2) We develop the *PIM-Quantifier* architecture and circuit, based on emerging non-volatile Spin-Orbit Torque Magnetic Random Access Memory (SOT-MRAM), optimized for our proposed mRNA quantification algorithm with fast and efficient one-cycle parallel XNOR&AND logic operations.
(3) We propose a large gene data partition and mapping algorithm to efficiently deploy the proposed mRNA quantification algorithm into our PIM-Quantifier hardware platform, which shows great increase in parallelism and throughput.
(4) We extensively assess our PIM-Quantifier with other recent non-volatile PIM platforms and software implementation (i.e. CPU) in performance and energy efficiency.

II. mRNA Quantification-in-Memory Algorithm

Our proposed mRNA quantification-in-memory algorithm is shown in Algorithm 1 and Fig.2, which requires following steps. First, each gene will be transferred to an index-table which contains two parts: *k-mers* and *k-comp* (line-1 in algorithm 1). All transcripts' sequences of a gene is fragmented into k-length substrings, defined as 'k-mer', starting from each position; and for each k-mer, the k-compatibility ('k-comp') classes are defined according to its presence in the transcript ('1' means present', whereas '0' means 'not present'). The k-comp classes are represented as a one-dimensional vector with '0's and '1's with size same as the number of transcripts

Algorithm 1 mRNA quantification

```
1: Generate index_table for each gene. Index table consists k-mers and associated
   k_comp classes.
2: Initialize result = ones(m,n) //m is the number of genes, n is the length of k-comp
3: input_fragment = short_read[i=0:j=k] // k is the length of k-mer
4: for input_fragment in short_read do
5:     for index_num < length(index_table) do
6:         if input_fragment in index_table{index_num} then
7:             k_comp = index_table{index_num}(input_fragment)
8:             result(index_num,:) = AND(result(index_num,:), k_comp)
9:         else
10:            result(index_num,:) = zeros(1,n)
11:        end if
12:    end for
13:    input_fragment = short_read[i+1:j+1]
14: end for
15: Return result //result indicates the compatible transcripts of all genes
```

in that specific gene. The k-mers along with its k-comp classes are then pushed into a Hashmap. The index-table is then constructed from the k-mers and their k-comp classes. The size of index-table depends on the value of k, and its size can be at most $L - k + 1$, for a gene with length L. Typically, human genome contains thousands of genes, each gene represented by one index-table which has thousands of k-mers on average, with k-mer length of 40. Note that, this index-table construction step is one-time effort for every gene, and it will be pre-generated and stored in our PIM platform.

Second, a sliding window with length same as k-mers will be used to generate input fragments for every short read (line-13). Each fragment will be sent to every gene index-table to search if an exact match (implemented using bit-wise XNOR logic in hardware) will be identified. Once the exact match in one index-table is found, the corresponding k-comp value will be recorded for the next step (line-7). If there is no match, it means this short read doesn't belong to any transcript in this gene. Thus, the whole short read should be discarded for this index-table (line-10). When all fragments are processed, the corresponding k-comps will be collected to conduct bit-wise AND operations (line-8). The value-'1' and its position in AND logic outputs indicate the corresponding transcript is compatible with current input short read.

To better explain the process, one example is shown in Fig.2. Each gene generates an index-table with the parameter "k-mer length: T" in the pre-computing stage. In this example, k-mer length is 3. It is also assumed this gene has 3 transcripts, resulting in the length of k-comp is 3. For example, the k-mer "AAC" has the corresponding k-comp "101", which means it is occurred in the transcript-0 and transcript-2 in this gene, but not in transcript-1. Again, this index-table only belongs to one gene, which is generated only once in advance and will be continuously used for processing new incoming input short reads. As an example in Fig.2, if the input short read length is 8, 6 k-mers with k=3 will be generated. Each fragment will be fed into the pre-generated index-table to find the exact match and its corresponding k-comp. Bit-wise AND operation is then performed on all the selected k-comps to produce the final output. In this example, based on the final AND output-'100', the transcript-0 is the found compatible transcript. Of course, the final output may have multiple '1's, indicating more than one transcript is compatible. We validated

our quantification-in-memory algorithm with existing software programs in [1], [2], showing the same computation results and similar computing complexity, while ours is optimized for PIM acceleration.

To summarize, for PIM hardware implementation, the main operations of our quantification-in-memory are k-mer matching (based on XNOR) and AND logic for matched k-comps. For k-mer matching, one of the XNOR based match operand is fixed (i.e. pre-computed k-mers in the index-table), and the other operand is fragment of input short read (i.e. a variable). This XNOR operation naturally matches with non-volatile PIM platform due to its greatly reduced leakage, non-volatility and parallel logic computation. Moreover, the matching operation among different index-tables are independent, where each computational array could be used as one matching engine to fully leverage the parallelism of PIM architecture. For AND operation, since it obeys the associative law, we will divide the whole bit-wise AND operations into consecutive AND2 logic operations. Therefore, for each input fragment, after XNOR matching to identify k-mer in each index-table, the corresponding k-comp will be activated to conduct AND logic with previous AND output, updating final output. Above analysis clearly shows that fast and parallel XNOR/AND logic operations are essential for PIM acceleration of quantification.

III. PIM-Quantifier Architecture and Circuit

Our proposed *PIM-Quantifier* 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(a). The main memory is composed of a set of MRAM chips. Each chip 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 Spin-Orbit Torque Magnetic Random Access Memory (SOT-MRAM) arrays as demonstrated in Fig.3(a)-(b). Every compute array includes two crucial sub-arrays termed as *K-mer* and *K-comp* arrays. They could work in two modes (i.e. memory and in-memory computing mode) to process the computationally-intensive bit-wise XNOR and AND logic, respectively, required by the quantification-in-memory algorithm. These two arrays stores different types of data, but using the same designs of memory row/column decoder, Sense Amplifier (SA) (Fig.3c(A)), write driver (Fig.3c(F)), and local row buffers (Fig.3c(E)). Fig.3b(C) shows the k-mer array architecture with a sample 3×3 array. Each SOT-MRAM cell is associated with the Write Word Line (WWL), Read Word Line (RWL), Write Bit Line (WBL), Read Bit Line (RBL), and Source Line (SL) to perform typical memory and in-memory computing operations. To program free-layer magnetization direction (thus low or high resistance level representing data - '0' and '1') of SOT-MRAM, flow of charge current (±y) through Spin Hall Metal-SHM (Tungsten, $\beta - W$ [12]) will cause accumulation of opposite directed electron spin on both surfaces of SHM due to spin Hall effect [13]. Thus, a spin current flowing in ±z is generated and further produces spin-orbit torque (SOT) on the adjacent free magnetic layer, causing switch of magnetization, as well as the resistance of SOT-MRAM cell (i.e. writing data).

To perform memory read and PIM logic operations, we propose to add a 2:1 MUX and a reference resistor (R_s) to each RBL, as shown in Fig.3c(A). For the typical memory read (e. g. $M1$), a read voltage is applied through the MUX's first input (V_1) to RBL1 and the sense current I_{sense} flows from the selected SOT-MRAM cell's resistance (R_{M1}) to ground. Then, assuming R_{M1} and R_s as two elements of a voltage divider, our voltage-based sensing mechanism generates $V_{sense} \simeq \frac{R_{M1}}{R_{M1}+R_s} \times V_{h/l}$ at the input of SA. This voltage is then compared with the memory mode reference voltage ($V_{sense,P} < V_{ref} < V_{sense,AP}$). Now, if the V_{sense} is higher (/lower) than V_{ref}, i.e. R_{AP} ($/R_P$), then the output of the SA produces High (/Low) voltage indicating logic '1' (/ '0'). In the computing mode, we propose to store the first operand in the memory as a resistance state where the second operand ('0'/'1') could be fed into the 2:1 MUX and selected by the ctrl unit. This will effectively convert the binary input into a proportional sense voltage (V_l/V_h) to drive the RBL. In this way, our voltage-based sensing mechanism generates the corresponding V_{sense} to various input combinations. Through selecting different reference voltages (En_{AND}, En_{OR}), the SA executes basic Boolean logic functions (i.e. AND and OR). For AND operation, V_{ref} is set at the midpoint of $V_{AP}//V_P$ ('1','0') and $V_{AP}//V_{AP}$ ('1','1'). In the k-mer array, by activating two enables (En_{AND}, En_{OR}) simultaneously for all the RBLs, bulk bit-wise XNOR2 could be implemented in a single memory cycle quite efficiently. Fig.3b(D) represents the k-comp array developed to handle the consecutive AND operation of the selected k-comp, leveraging the same logic-in-memory design. The all-zero detection circuit in Fig.3c(B), as explained in algorithm section, is used to detect whether XNOR output is all zero (need to discard current short read). Fig.3c(E) is the shift register to generate the fragment from input short read.

IV. Mapping to PIM-Quantifier

In this section, we present how to deploy the mRNA quantification to PIM-Quantifier. To start, each pre-computed index-table will be stored in compute array consisting of k-mer array and k-comp array. Both k-mers and k-comps are stored along bit-lines required by the property of above discussed in-memory-logic designs and friendly for parallel computing. However, the k-mer table size could be very large, making it difficult to fit into one memory sub-array. Thus, we introduce an index-table partition method with property that k-mers within the same memory sub-array share the same one or more front-end nucleotides (*nt*) depending on the total data size and memory sub-array size. The advantage of such partition method is that it could save several XNOR cycles for the front-end nt(s). For example, in Fig.4, the k-mers in sub-array-1 are all starting with nt-'A'. Note that, this rule could be partially relaxed if k-mers starting with different nt could be all stored in the same memory sub-array. When the input fragment is

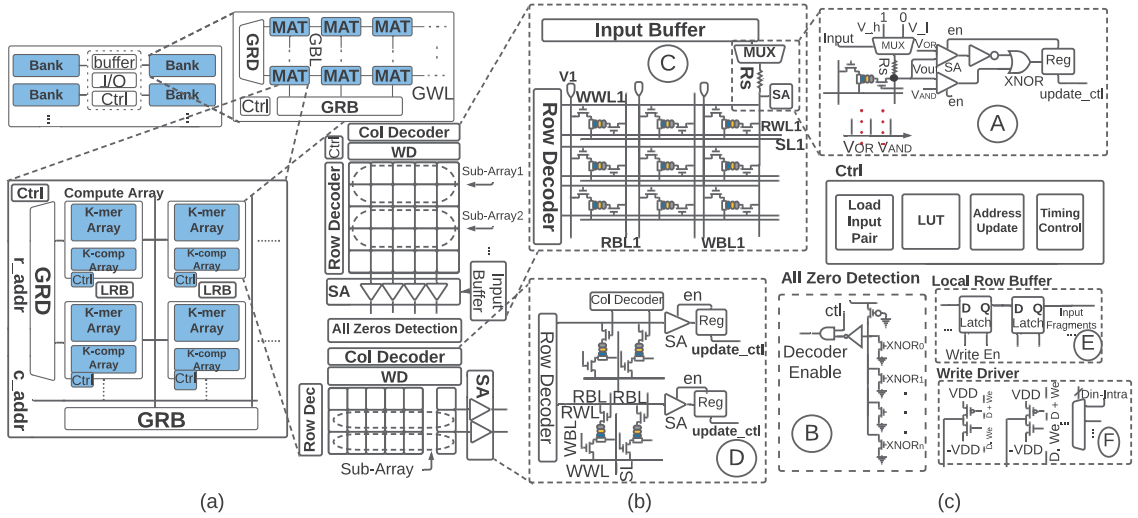
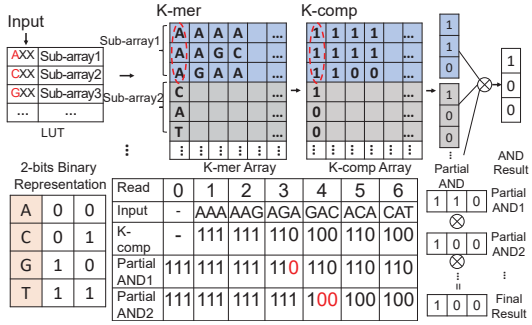


Fig. 3. (a) PIM-Quantifier architecture, (b) SOT-MRAM based computational array contains both k-mer and k-comp array, (c) Peripheral circuitry.



Read	0	1	2	3	4	5	6
Input	-	AAA	AAG	AGA	GAC	ACA	CAT
K-comp	-	111	111	110	100	110	100
Partial AND1	111	111	111	110	110	110	110
Partial AND2	111	111	111	111	100	100	100

A	0	0
C	0	1
G	1	0
T	1	1

Fig. 4. mRNA quantification-in-memory process and data mapping.

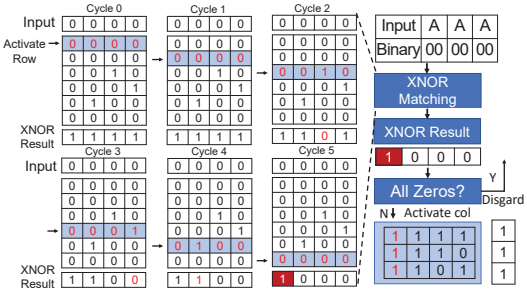


Fig. 5. Parallel search&matching operation

received, the first step is to locate which memory sub-array should be directed for the next matching stage, which could be implemented by a small look up table (LUT).

In Fig.4, assuming the input fragment is 'AAA'. According to the LUT, the k-mers staring with 'A' are all stored in the sub-array-1. Thus we activate the sub-array-1 for the XNOR based matching operation as discussed in previous section. After that, it identifies one match, indicating 'AAA' is stored in the first column in the sub-array-1. Thus, the corresponding k-comp value stored in the first column -'111' will be activated to conduct AND logic with the previous partial AND result stored in the latch. After this round of AND operation, its result will be saved into the latch to update partial AND result. When all fragments of a short read are processed, all the partial AND results from each sub-array stored in their latches will be collected to conduct final round of AND operation to generate the final output, indicating which transcript is compatible with current input short read. Fig.4 provides an example to process 6 input fragments.

For the XNOR based matching Within sub-array, we use the first input fragment 'AAA' as one input example as shown in Fig.5. Since there are only 4 types of nt, we use two bits to encode them defined in Fig.4. Thus, the input 'AAA' is encoded as '000000'. It needs 6 cycles to perform XNOR based matching within the corresponding sub-array. As mentioned earlier, the k-mers are stored along bit-lines. In this 6x4 example array, it shows 4 k-mers, i.e. 'AAA','AAG', 'AGA', and 'ACA', from left to right. To maximize computing parallelism, multiple bit-lines (4 in this example) will be activated at the same time to conduct parallel XNOR logic between the input and stored k-mers bit by bit. First, it will check if all k-mers in this array are staring with 'A'. According to the XNOR result, it excludes those k-mers that are not staring with 'A', to narrow the search space for next nt. After two nt (i.e. 4 bits) matching, the XNOR based match result are '1100', indicating the corresponding first two k-mers are matched up to now, i.e. 'AAA' and 'AAG'. Similar XNOR based match will compare the input with the last nt, generating XNOR based match result as '1000'. It indicates the first '1' is matched with input-'AAA'. Then, the first k-comp in the corresponding k-comp array is activated for the following AND operation. Of course, it is possible there is no exact match in this k-mer array. In that case, the XNOR based match result should be all-zeros, which will be detected based on the circuit shown in Fig.3(c)(B). Correspondingly, no k-comp will be activated for the following AND operations.

V. Performance Estimation

A. Experiment Setup

To assess the performance of PIM-Quantifier as the new PIM platform from circuit-level up to algorithm-level, we develop a cross-layer comprehensive simulator similar as [10]. The PIM-Quantifier's sub-array and peripheral circuits are

designed in Cadence Virtuoso with 45nm NCSU Product Development Kit (PDK) library [14] and then evaluated in Cadence Spectre for the circuit-level performance parameters. The architecture-level simulator is based on NVSim [15] where the configuration file is flexible and corresponding to different array design and working mechanism. Thus, different types of PIM platforms can share the similar organization and simulator for fare comparison. For those Content Addressable Memory (CAM) based designs, we use NVsim-CAM [16] to estimate their performances. On top of architecture simulator, we use Matlab to pre-process the real genome data. The cross-layer simulator could evaluate latency, energy, and throughput for the alignment-free based quantification with human genome hg38 dataset.

We use 1 million short reads with length of 101 as test inputs. 22000 genes (index-tables) are tested in total. Each index-table contains 3000 to 10000 k-mers with length of 25. We configure the PIM-Quantifier's memory array with 256 rows and 1024 columns, 8x2 mats (with 1/1 as row/column activation) per bank organized in H-tree routing manner, 64x64 banks (with 1/1 as row/column activation) in each memory group. Totally, 65K sub-arrays are enough in most cases.

In the rest of this section, we first analyze the bulk bit-wise operations for the proposed platform. The Monte-Carlo simulation is also performed to show its stability. Then, more detailed experiments are conducted to include different PIM hardware platforms comparison, data-mapping optimization, and real gene data.

B. Circuit Level Analysis

Functionality. Fig.6 depicts the transient simulation results of a single k-mer/k-comp sub-array based on the architecture shown in Fig.3a. For the sake of clarity, we assume a 3ns period clock synchronises the write and read operations. However, a 2ns clock period could be used for a reliable read operation. During the precharge phase of SA (Clk=1), the V_{write} voltage is set and applied to the WBL to change the selected SOT-MRAM cell's resistance to R_{low}= 5.9$k\Omega$ or R_{high}=15.7$k\Omega$. This way, the first operand is stored into the memory bit-cell as a resistance state. Prior to the evaluation phase (Eval.) of SA, WWL and WBL is grounded. The second operand ('0'/'1') is converted to a sense voltage (400mV/500mV) and fed to the RBL. In the evaluation phase, RWL goes high. Depending on the resistance state of SOT-MRAM bit-cell, V_{sense} is generated through the resisitve voltage divider with the R_s= 5$k\Omega$ as the first input of SA, when V_{ref} is applied at the second input of SA. The comparison between V_{sense} and V_{ref} for all possible input cases are plotted in Fig.6. We observe when V_{sense}<V_{OR} (only in the first evaluation phase), the SA outputs binary '0', whereas output is "1". We also plotted the I_{sense} to analyze possible read disturbance when applying the V_{sense}. It can be seen that in the worst case I_{sense} ($15\mu A$)$\ll$ I_{write} ($130\mu A$).

Variation analysis. To validate the variation tolerance of the sensing circuit, we have performed a worst-case scenario Monte-Carlo simulation with 100000 trials. A $\sigma = 5\%$ variation is added to the Resistance-Area product (RA_P), and a

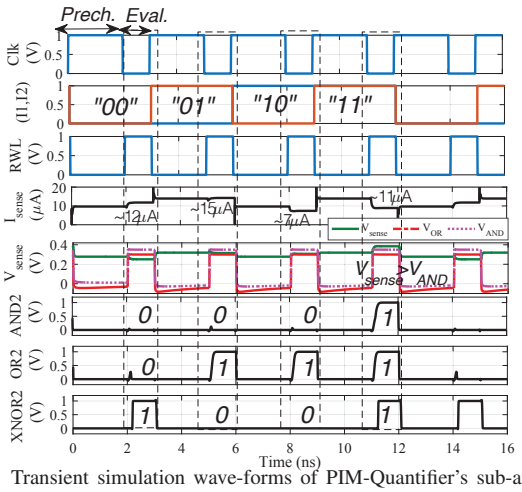


Fig. 6. Transient simulation wave-forms of PIM-Quantifier's sub-array and its reconfigurable SA for performing single-cycle in-memory operations.

$\sigma = 10\%$ process variation (typical MTJ conductance variation [17]) is added on the TMR. The simulation result of sense voltage (V_{sense}) distributions for the presented one-row activation in-memory mechanism is shown in Fig.7a. We observe that 34.2mv and 18.7mv sensing margin are achieved between three possible cases. Fig.7b shows the sensing margin for the conventional 2-row activation PIM logic. It can be seen that the presented design provides larger sensing margins especially when it comes to "01" and "11" margin. This is mainly due to the fact that, assuming R_{M1} and R_{M2} as two MRAM cells located in a same bit-line and R_s as the reference resistor, our voltage-based sensing mechanism provides $V_{sense} \simeq \frac{R_{M1}}{R_{M1}+R_s} \times V_{h/l}$, where the current-based two-row activation mechanism [18] provides $V_{sense} \simeq I_{sense} \times (R_{M1} // R_{M2})$. Since the parallel resistance is virtually half of the resistance of a single cell.

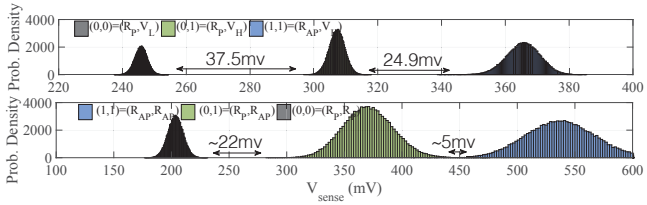


Fig. 7. Monte-Carlo simulation of V_{sense} (with RA_P/TMR=5%/10% - t_{ox}=1.2nm) for (a) one-row activation PIM scheme (b) conventional two-row activation PIM scheme.

C. Experiment Results

Since there is no prior PIM based hardware acceleration of mRNA quantification, to conduct fair comparison, we re-implement several representative non-volatile PIM designs from CAM [20]–[22], IMCE [18], Pinatubo [19], to also deploy our quantification-in-memory algorithm in those platforms. Similar to our design, the CAM based platforms is configured to only store one XNOR operand in memory array and convert the other operand as voltage/current input. Thus, those CAM based platforms share the same memory structure like our design. The IMCE and Pinatubo need to write both XNOR operands into the non-volatile memory array for logic functions, thus larger memory array size is needed for theses two platforms to use the same index table

TABLE I
ENERGY AND LATENCY OF DIFFERENT PLATFORMS

	PIM-Quantifier	IMCE [18] 512x1024	Pinatubo [19] 512x1024	RRAM CAM [20]	MRAM CAM [21]	PCM CAM [22]
Technology(nm)	45nm	45nm	45nm	45nm	45nm	45nm
Latency(ns)	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)	90.94pJ	135.940pJ	137.436pJ	54.43pJ	697.28pJ	116.7pJ
Energy(Write/row pJ)	61.34pJ	92.092pJ	1.088nJ	1.200nJ	147.96pJ	7.34nJ

partition. But the MAT and bank organization remains the same as other platforms. We also compare the CPU (Intel E5-2620) performance using state-of-the-art mRNA transcript quantification software-Kallisto [1].

Table.I, Fig.8 and Fig.9 summarize the key performance of different PIM platforms and CPU. Thanks to our optimized circuit designs, PIM-Quantifier requires less read&write energy than IMCE and Pinatubo. Different from IMCE and Pinatubo, the input of PIM-Quantifier does not need to be written into memory array for computing, eliminating extra input operand writing power/latency and smaller memory size required. As expected, CAM based platforms achieve highest throughput due to its high parallel matching scheme. But such high throughput is at the cost of extremely high power consumption. Moreover, those platforms need 4T2R cell structure, making each memory cell two times larger than other PIM platforms. If defining the efficiency as 'throughput/power', as shown in Fig.8(c), PIM-Quantifier is 1.7x - 71.5x more efficient (Throughput/Power) than other PIM based platforms. CPU has the worst efficiency in this case, almost three orders worse than all PIM based platforms. Fig.9 shows the normalized comparison of throughput/power/area of different PIM platforms, showing PIM-Quantifier greatly outperforms others.

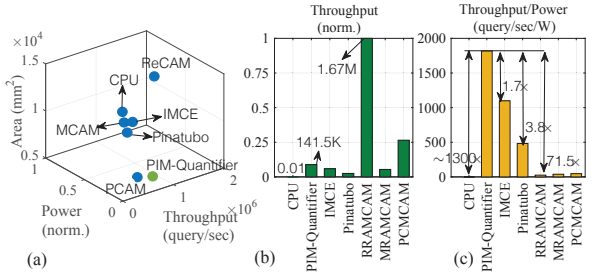


Fig. 8. (a) Comparison between area, power, and throughput of different PIM accelerators and CPU. (b) Normalized throughput and (c) Throughput/Watt.

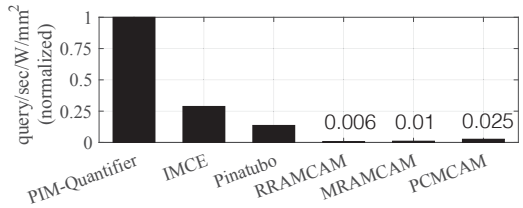


Fig. 9. Normalized throughput/power/area of various PIM platforms.

VI. CONCLUSION

In this work, we present PIM-Quantifier to accelerate compute- and data-intensive mRNA quantification. A PIM-friendly quantification algorithm is presented along with the optimized hardware platform and mapping method. The experiments show that mRNA quantification on PIM platforms with our algorithm has orders higher throughput than traditional CPU implementation. PIM-Quantifier also achieves the best efficiency, defined as throughput/watt, among recent PIM platforms compatible with our quantification-in-memory algorithm.

REFERENCES

[1] N. L. Bray *et al.*, "Near-optimal probabilistic rna-seq quantification," *Nature biotechnology*, vol. 34, no. 5, pp. 525–527, 2016.

[2] R. Patro *et al.*, "Salmon provides fast and bias-aware quantification of transcript expression," *Nature methods*, vol. 14, no. 4, 2017.

[3] A. Dobin *et al.*, "Comment on "tophat2: accurate alignment of transcriptomes in the presence of insertions, deletions and gene fusions" by kim et al." *Biorxiv*, p. 000851, 2013.

[4] C. Trapnell *et al.*, "Transcript assembly and quantification by rna-seq reveals unannotated transcripts and isoform switching during cell differentiation," *Nature biotechnology*, vol. 28, no. 5, pp. 511–515, 2010.

[5] P. Siegl *et al.*, "Data-centric computing frontiers: A survey on processing-in-memory," in *Proceedings of the 2nd International Symposium on Memory Systems*, ser. MEMSYS '16, 2016, p. 295–308.

[6] M. Hu *et al.*, "Dot-product engine for neuromorphic computing: Programming 1t1m crossbar to accelerate matrix-vector multiplication," in *53rd DAC*, ser. DAC '16, 2016.

[7] K. Kim *et al.*, "An energy-efficient processing-in-memory architecture for long short term memory in spin orbit torque mram," in *ICCAD*, 2019, pp. 1–8.

[8] S. Angizi *et al.*, "Pim-assembler: A processing-in-memory platform for genome assembly," in *57th DAC*, 2020, pp. 1–6.

[9] Z. I. Chowdhury *et al.*, "A dna read alignment accelerator based on computational ram," *IEEE JXCDC*, vol. 6, no. 1, pp. 80–88, 2020.

[10] S. Angizi *et al.*, "Aligns: A processing-in-memory accelerator for dna short read alignment leveraging sot-mram," in *56th DAC*, 2019, pp. 1–6.

[11] F. Zokaee *et al.*, "Aligner: A process-in-memory architecture for short read alignment in rerams," *IEEE Computer Architecture Letters*, vol. 17, no. 2, pp. 237–240, 2018.

[12] C.-F. Pai *et al.*, "Spin transfer torque devices utilizing the giant spin hall effect of tungsten," *Applied Physics Letters*, vol. 101, no. 12, 2012.

[13] X. Fong *et al.*, "Spin-transfer torque devices for logic and memory: Prospects and perspectives," *IEEE TCAD*, vol. 35, no. 1, pp. 1–22, 2016.

[14] *Ncsu eda freepdk45. [Online].*, http://www.eda.ncsu.edu/wiki/FreePDK45.

[15] X. Dong *et al.*, "Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *IEEE TCAD*, vol. 31, no. 7, pp. 994–1007, 2012.

[16] S. L. andothers, "Nvsim-cam: A circuit-level simulator for emerging nonvolatile memory based content-addressable memory," in *2016 ICCAD*, 2016, pp. 1–7.

[17] X. Fong *et al.*, "Spin-transfer torque memories: Devices, circuits, and systems," *Proceedings of the IEEE*, vol. 104, no. 7, 2016.

[18] S. Angizi *et al.*, "Imce: Energy-efficient bit-wise in-memory convolution engine for deep neural network," in *23rd ASP-DAC*, 2018, pp. 111–116.

[19] S. Li *et al.*, "Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories," in *53rd DAC*, 2016, pp. 1–6.

[20] Li-Yue Huang *et al.*, "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*, 2014, pp. 1–2.

[21] S. Matsunaga *et al.*, "A 3.14 um2 4t-2mtj-cell fully parallel tcam based on nonvolatile logic-in-memory architecture," in *VLSIC*, 2012.

[22] J. Li *et al.*, "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*, vol. 49, no. 4, pp. 896–907, 2014.