Aligner-D: Leveraging In-DRAM Computing to Accelerate DNA Short Read Alignment

Fan Zhang, *Graduate Student Member, IEEE*, Shaahin Angizi, *Senior Member, IEEE*, Jiao Sun, Wei Zhang, and Deliang Fan, *Member, IEEE*

***Abstract*— DNA short read alignment task has become a major sequential bottleneck to humongous amounts of data generated by next-generation sequencing platforms. In this paper, an energy-efficient and high-throughput Processing-in-Memory (PIM) accelerator based on DRAM (named *Aligner-D*) is presented to execute DNA short-read alignment with the state-of-the-art BWT alignment algorithm. We first present the PIM design that utilizes DRAM's internal high parallelism and throughput. It converts each DRAM array to a potent processing unit for alignment tasks. The proposed *Aligner-D* can efficiently execute the bulk bit-wise XNOR-based matching operation required by the alignment task with only 3-transistor/col overhead. We then introduce a highly parallel and customized read alignment algorithm based on BWT that supports both *exact* and *inexact* match tasks. Next, we present how to map the correlated data of the alignment task to utilize the parallelism from both new hardware and algorithm maximumly. The experimental results demonstrate that *Aligner-D* obtains ∼4×, ∼2.45×, ∼3.26×, and ∼1.65× improvement, respectively, compared with other in-memory computing platforms: Ambit (Seshadri et al., 2017), DRISA-1T1C (Li et al., 2017), DRISA-3T1C (Li et al., 2017), and ReDRAM (Angizi and Fan, 2019). As for DNA short read alignment, *Aligner-D* boosts the alignment throughput per Watt by ∼20104×, ∼3522×, ∼927×, ∼88×, ∼5.28×, and ∼2.34×, over ReCAM, CPU, GPU, FPGA, Ambit, and DRISA, respectively.**

***Index Terms*— DNA short read alignment, processing-in-memory, DRAM, accelerator.**

I. Introduction

THE novel DNA sequencing method, on top of the recent high-throughput genomic technologies, is able to analyze the accurate order of nucleotides (*nt*) along genomes and measure cells' molecular activities. Such advances improve diagnostics of disease and different aspects of medical care, such as prenatal testing and tailoring patient treatment [3], [4]. In general, the generated sequence data of one patient sample is composed of tens of millions of short DNA sequences (short reads) ranging from 50-500 nucleotide-*nt* in length with no position information. Thus, it is required to determine what part of the chromosome/genome they are from before most genomic analyses can start. This is achieved by aligning the short reads to the reference genome. The reference genome contains two paired twisting strands where each strand consists of roughly 3.2 billion *nt* bases (A, T, C, G) in human specifically paired as A-T and C-G [5], [6]. As a result, for a single sample, the DNA short read alignment task is to map the reads (tens of millions) to a reference genome (3.2 billion base pair-*bp*) allowing 1-2 mismatches on each short read. Various alignment algorithms have been developed during the last decade. However, even the efficient algorithms such as Bowtie [7] or BWA [8] based on Burrows-Wheeler Transformation (BWT) seek hours or even days to align the short reads generated by one run (Terabytes of DNA sequence data) from DNA sequencing machine. Therefore, the genomic information achieved from DNA sequencing data cannot be applied for prognosis or disease diagnosis in clinics and hospitals.

Today's sequencing acceleration solutions including CPU, GPU [9], ASIC [4], [10], [11], and FPGA [12] are mostly based on the Von-Neumann architecture that unavoidably consumes a large amount of energy in data movement due to the so-called "memory wall" challenge, as it has separate memory and processing components connecting through buses. The data movement between memory and processor consumes even more energy than the computation itself in such data-intensive applications [13]. Meanwhile, Processing-in-Memory (PIM) architecture has been actively investigated for different data-intensive applications [1], [2], [14] as a potential way to overcome the "memory wall" challenge. The main idea of PIM is to incorporate logic computation within memory units such that memory is able to process data internally. Such platforms are expected to provide inherent parallel processing mechanisms exploiting large internal memory bandwidth. PIM architecture could be implemented with either non-volatile or volatile memories. Non-volatile memories(NVMs) are usually implemented as Vector-Matrix Multiplication(VMM) engine through analog computing [13], [14], [15], [16]. There are also attempts to use NVMs to realize the logic computation [17], [18]. But these works need a lot of writing to NVMs, which conflicts with NVM's limited endurance and high writing energy properties [13], [19], [20]. In the meanwhile, the processing-in-SRAM platforms have been developed in

Manuscript received 30 September 2022; revised 9 January 2023; accepted 24 January 2023. Date of publication 1 February 2023; date of current version 20 March 2023. This work was supported in part by the National Science Foundation under Grant 2003749, Grant 2144751, Grant 2153525, and Grant 2228028. This article was recommended by Guest Editor M. H. Najafi. *(Corresponding author: Deliang Fan.)*

Fan Zhang and Deliang Fan are with the School of Electrical, Computer and Energy Engineering, Arizona State University, Tempe, AZ 85281 USA (e-mail: fzhang95@asu.edu; dfan@asu.edu).

Shaahin Angizi is with the Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, NJ 07102 USA (e-mail: shaahin.angizi@njit.edu).

Jiao Sun and Wei Zhang are with the Department of Computer Science, University of Central Florida, Orlando, FL 32816 USA (e-mail: wzhang.cs@ucf.edu).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/JETCAS.2023.3241545.

Digital Object Identifier 10.1109/JETCAS.2023.3241545

recent literature [21], [22] due to SRAM's mature process and universal application. Considering the larger memory capacities of DRAM and off-chip data movement reduction as opposed to SRAM-based PIM, processing-in-DRAM platforms [1], [2], [3], [23], [24], [25], [26]) have also gained a lot of attention. Therefore, with significantly higher memory capacity and throughput in performing bulk bit-wise operations by either changing the sub-array's sense amplifier and/or DRAM cell, the processing-in-DRAM platform is becoming a promising accelerator for different data-intensive applications. For example, Ambit [1] presents a triple-row activation technique to carry out a majority-based AND/OR logic, excelling NVIDIA GeForce GPU, and even HMC [27], respectively by 32.0×, and 2.4×. For accelerating convolutional neural networks, DRISA [2] presents two alternative 3T1C- and 1T1C-based PIM techniques and improves speedup and energy-efficiency by 7.7× and 15× against GPUs. Nevertheless, when it comes to XNOR- and addition-based tasks, such as data encryption and DNA alignment, there exist multiple challenges that make these platforms inefficient acceleration solutions. This comes from the intrinsic logic complexity of XNOR2 operation. i.e., the Ambit, DRISA-3T1C, and DRISA-1T1C need 7, 4, 2 cycles to complete the XNOR2 operation, respectively, with a significant area/power overhead. Therefore, the existing processing-in-DRAM platforms are not able to offer a high-throughput XNOR-based operation despite leveraging the memory level parallelism and maximum internal bandwidth.

To tackle the memory bandwidth bottleneck and address the existing challenges, in this work, we present an energy-efficient and high throughput processing-in-DRAM accelerator, named *Aligner-D*. *Aligner-D* uses a novel processing-in-memory circuit that only needs three transistors to execute bulk bit-wise XNOR2 operations between one stored operand from the memory array and one feed input. With no change in the sense amplifier circuit and DRAM cell, this new technique is designed on top of the traditional DRAM. The XNOR2 operation is realized efficiently on every vertical memory bit-line. In addition, such a design addresses the multi-cycle operations of the prior methods discussed earlier and the reliability concerns regarding the bit-line voltage deviation. We assess and compare *Aligner-D*'s performance with different computing systems including Core-i7 Intel CPU [28], NVIDIA GTX 1080Ti Pascal GPU [29], HMC 2.0 [27], Ambit [1], DRISA [2], etc., to perform bit-wise tasks. We observe that *Aligner-D* achieves a considerably higher throughput as opposed to Von-Neumann computing systems, CPU/GPU. Besides, *Aligner-D* outperforms other PIMs in performing XNOR-based operations by ∼4×, ∼2.45×, ∼3.26×, and ∼1.65×, respectively, compared with Ambit [1], DRISA-1T1C [2], DRISA-3T1C [2], and ReDRAM [3]. From the energy consumption perspective, *Aligner-D* reduces the DRAM energy by ∼ 23%, 16%, 33.8%, and 22.7%, respectively, compared with Ambit [1], DRISA-1T1C [2], DRISA-3T1C [2], and ReDRAM [3] and ∼51× compared to data transfer via the DDR4 interface. We further assess *Aligner-D*'s efficacy by executing a new DNA alignment algorithm to accelerate intensive matching and addition operations in both exact and inexact matches. We observe that *Aligner-D* increases the read alignment throughput per Watt by ∼3522×, ∼927×, ∼88×, ∼5.28×, and ∼2.34×, over CPU, GPU, FPGA, Ambit, and DRISA, respectively. The main contributions of this work are listed as follows:

- We propose an energy-efficient, high-throughput, and XNOR-friendly PIM architecture *Aligner-D* with a set of new circuit-level and microarchitectural methods to implement a data-parallel computational core for data-intensive applications;
- We investigate the optimization of an FM-index-based DNA short sequence alignment algorithm with BWT to fully leverage *Aligner-D*'s parallelism and boost alignment task speed. We develop a new data partitioning and mapping technique for locally handling the indices supporting different lengths of DNA reads;
- We extensively assess and compare *Aligner-D*'s performance with current DNA short read alignment acceleration solutions such as CPU, GPU, ASIC, FPGA, etc.

The remainder of the paper is presented as follows. Section II discusses the state-of-the-art processing-in-DRAM designs, BWT read mapping, and DRAM-based read mapping challenges. Section III delineates our proposed accelerator design, i.e., *Aligner-D*, and the supported in-memory operations with performance analysis. Section IV presents our customized *Aligner-D* sequencing algorithm for exact and inexact DNA matching. Section V shows the correlated hardware mapping method and computation in *Aligner-D*. Section VI gives the simulation results. Section VII proposes the potential challenges and the future work. Finally, Section VIII concludes this work.

II. Background and Motivation

A. Processing-in-DRAM Designs

At the top architectural level, a DRAM hierarchy includes channels, modules, and ranks. With a typically 64-bits wide data bus, each memory rank consists of multiple memory chips. The memory chips are designed with various configurations and operate simultaneously [1], [30], [31]. The memory chip is spilt into several memory banks. Each bank is composed of 2D sub-arrays of memory bit-cells that are virtually-ordered in memory matrices (mats). Banks located in the same chip typically share buffer and I/O and banks located in different chips work in a lock-step manner. As depicted in Fig. 1a, the memory sub-array consists of 1) Memory rows (normally 2^9 or 2^{10}) connected to DRAM cells, 2) A Sense Amplifiers' (SA) row, and 3) A memory Row Decoder (RD) connected to the word-lines. Structurally, a DRAM cell is composed of two modules, a storage module (capacitor) and an access module (Access Transistor-AT) as shown in Fig. 1b. The gate and drain of DRAM's AT are connected to the Word-line (WL) and Bit-line (BL), respectively. DRAM cell stores the binary data by the charge of the capacitor. It encodes a fully-charged (V_{dd}) capacitor as logic '1' and no-charge capacitor as logic '0'.

1) Read/Write Operation: For read/write operation, both BL and $\overline{BL}$ are initially pulled to $\frac{V_{dd}}{2}$. Technically, accessing data from a DRAM's sub-array after the initial state is accomplished with three commands [1], [32] by the memory controller: 1) With the activation command (i.e. ACTIVATE),

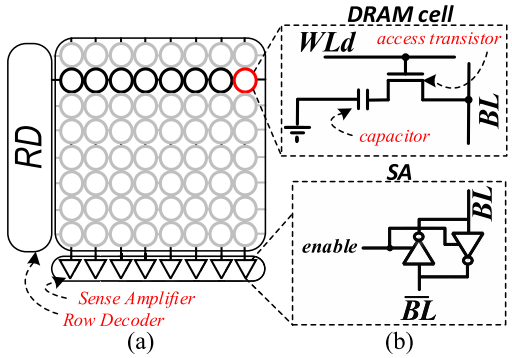


Fig. 1. (a) The organization of a DRAM sub-array, (b) DRAM cell structure and Sense Amplifier.

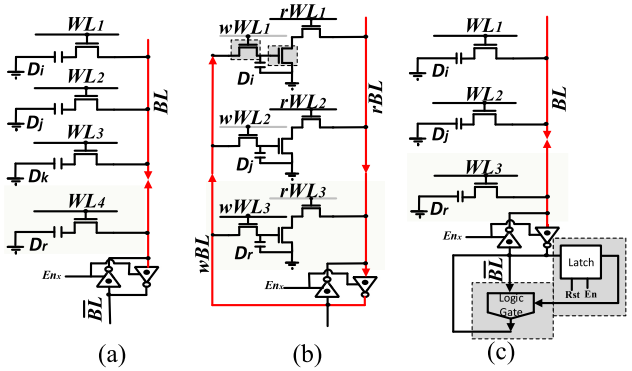


Fig. 2. (a) TRA mechanism in Ambit [1], (b) 3T1C mechanism in DRISA [2], (c) 1T1C-logic mechanism in DRISA [2]. Glossary- D_i/D_j: input rows data, D_k: initialized row data, D_r result row data.

a target row is activated and stored row data is transferred from the DRAM row cells to the SA row. Fig. 1b depicts the connection between a cell and the SA via a BL. The selected cell typically shares its charge value ($0/V_{dd}$) with the BL which slightly changes the initial BL's voltage ($\frac{V_{dd}}{2} \pm \delta$). Then, the memory controller activates the *enable* signal that makes the SA amplify the δ towards the original value of the data through voltage amplification leveraging the switching threshold of SA's inverter [32]. 2) By a `WRITE/READ` command, the data can be then moved to/from SA from/to DRAM bus. It is noteworthy that several `READ/WRITE` commands can be issued to one row. 3) With a `PRECHARGE` command, both BL and $\overline{BL}$ precharge again to the initial state and get ready for the next access cycle.

2) Initialization and Copy Operation: For a very fast in-memory copy operation ($< 100ns$) within DRAM sub-arrays, instead of $\sim 1\mu s$ copy operation in Von-Neumann computing architecture, *RowClone*-Fast Parallel Mode (FPM) [33] offers a new method that does not require sending the data to the processing units. In this method, issuing two back-to-back `ACTIVATE` commands (without `PRECHARGE` command in between) to the source and destination rows can realize a $90ns$ and multi-kilo byte in-memory copy operation. This technique has been further exploited for row initialization to effectively copy a preset DRAM row ('1' or '0') to a single or multiple destination row(s) incurring a 0.01% overhead to memory chip area [33].

3) Other Logic Functions: To implement the logic operation in processing-in-DRAM architecture, the RowClone idea was extended in the *Ambit* [1] to realize three-input majority-based operations (`Maj3`) in memory through simultaneously issuing the `ACTIVATE` command to three rows with a `PRECHARGE` command afterwards, named *Triple Row Activation (TRA)* mechanism. Ambit incurs just 1% area overhead to DRAM chip [1]. Having one row as the control (D_k), as illustrated in Fig. 2a, initialized by '0'/'1', TRA implements in-memory `AND2/OR2` based on `Maj3` function via charge sharing among connected DRAM cells (D_k, D_i and D_j) and writes the result back on D_r cell. In addition, Ambit employs the TRA method along with Dual-Contact Cell(DCC) to implement the complementary operations. Nevertheless, Ambit deals with multi-cycle PIM operations to realize other logic functions like `XOR2/XNOR2`. [34] further extends the idea of using charge sharing to realize the majority function among the data connected in the same bit-line. Eventually, the majority state '1'/'0' will pull up/down the bit-line voltage. By simply detecting the difference between BL and $\overline{BL}$, the SA can give the majority gate result without any other additional computational circuit. With the help of the $Maj5$ function, [34] also realizes the full adder in the same memory sub-array. The *DRISA*-3T1C mechanism [2] alternatively leverages the 3-transistor DRAM design [35]. As shown in Fig. 2b, such cell design is composed of two separated write/read ATs, and one additional transistor for decoupling the capacitor from the read BL (rBL). This additional transistor links the two input DRAM cells in a NOR style on the rBL to perform the Boolean-complete `NOR2` function. *DRISA*-3T1C incurs a large area overhead and needs multi-cycle operations to realize different logic functions. As depicted in Fig. 2c, *DRISA*-1T1C mechanism [2] performs in-memory operations via an upgraded SA consisting of a CMOS logic gate and a latch. This mechanism performs in-memory operations in two consecutive cycles: 1) reading out D_i and storing it in the latch as the first input of CMOS logic, and 2) reading out D_j as the second input to perform the computation. This method requires excessive cycles to realize other logic functions and imposes a minimum of 12 transistors to each SA. The *Dracc* [36] recently designs a carry look-ahead adder by improving Ambit [1] to accelerate convolutional neural networks.

B. BWT-Based DNA Short Read Alignment

Sequence alignment algorithms (e.g., BWA [8] and Bowtie [7]) take all the advantages of BWT and index the large reference genome S to implement the read alignment efficiently. The BWT is a reversible rearrangement of a character string. Exact alignment finds all occurrences of the short read R (m bp) in the reference genome S (n bp). Fig. 3 shows an intuitive example of the exact alignment of a sample read-$R = CTA$ to a sample reference $S = TGCTA\$$ extracted from a gene, in which \$ denotes the end of a sequence. BW matrix (constructed by lexicographically sorting the strings originated from circulating string S) makes the Suffix Array (S_A) of a reference genome S a lexicographically-sorted array of the suffixes of S, where each suffix is represented by its position in S. In this way, the last column in the BW matrix is the BWT of reference S, here, $BWT(S) = ATGTC\$$. The FM-Index is then built on top of BWT providing the occurrence information of each symbol in it. The S_A interval (low, $high$) covers a range of indices where the suffixes share the same prefix. A backward search of the matched positions in the

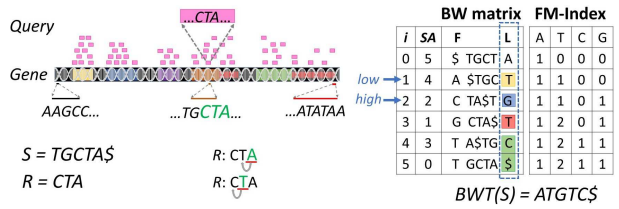


Fig. 3. Short read alignment concept.

reference genome S is then executed for each short read-R starting from the rightmost nucleotide (A in Fig. 3). The matched lower bound (low) and upper bound ($high$) in a S_A of the S for each nucleotide in R are determined based on FM-Index and count function [8]. Thus, the S_A interval can represent all the occurrences of the query string. At the end of the search, if $low < high$, R has found a match in S. Otherwise, it has failed to find a match. The complexity of this alignment algorithm is linearly proportional to the number of nucleotides in a read ($\mathcal{O}(m)$) in contrast to dynamic programming algorithms such as Smith-Waterman (SW) with $\mathcal{O}(nm)$ complexity [37]. Backtracking can simply extend the exact alignment algorithm to allow mismatches to support inexact alignment. In this approach, the DNA short read is permuted using edit operations (substitutions, insertions or deletions) [6], [7], [38], [39], [40], [41].

C. DRAM-Based Computing Challenges and Our Solutions Summary

In this work, we investigate the optimization of a fast FM-index-based DNA short read alignment algorithm and focus to leverage *Aligner-D*'s parallelism to accelerate it from a hardware perspective, as fully discussed in the next sections. To achieve this goal, two primary in-memory functions need to be designed and accelerated in the DRAM platform: 1) Matching based on `XNOR2` logic and 2) `Addition`. In this way, there are four main challenges that this work aims to resolve:

- **Limited throughput (Challenge-I):** Offering an area-efficient and high-throughput in-memory `XNOR2/addition` operation in PIM context is challenging due to the inherent complexity of XNOR-based logic implementations as discussed in recent PIM platforms (DRISA [2], Ambit [1], and Dracc [36]). Nevertheless, these platforms can utilize maximum memory-level parallelism and internal memory bandwidth to implement `NOT`, `(N)OR`, `(N)AND`, and `MAJ/MIN` functions. While the DRISA-1T1C mechanism is able to realize `XNOR2`, it still needs two consecutive cycles to handle the in-memory operation which in turn hinders other logic implementation's efficiency. We address this challenge by proposing the 3T XNOR and 12T full adder mechanism in Section III-A.
- **Row initialization (Challenge-II):** Considering R=AopB function ($op \in$ `OR2/AND2`), TRA mechanism [1], [32] requires at least four consecutive cycles to compute the output highly relying on the row initialization: 1-RowClone row A to row D_i (to avoid data-overwritten, first operand is copied to a computation row), 2-RowClone row B to D_j, 3-RowClone ctrl row to D_k (initialized row is copied to a computation row), 4-TRA and RowClone data of row D_i to R row (computation and writing-back the result). Therefore, the TRA mechanism requires an average of 360ns to execute. When it comes to `XOR2/XNOR2` operation, Ambit requires at least three row-initialization steps to process two input rows. Obviously, this row-initialization load could adversely impact the PIM's energy-efficiency, especially dealing with such big data problems. This challenge is addressed in Section III-A through the proposed 3T XNOR gate, which totally eliminates the need for initialization in performing XNOR-based logics.
- **Reliability concerns (Challenge-III):** By simultaneously activating 3 DRAM cells in the TRA mechanism, the voltage deviation on the BL could be less than the normal read operation. This will make the sense amplification stage longer or even adversely affect the reliability of the operation [1], [32]. For XNOR-based computations, the issue can even get worse since multiple TRAs are required. To address this challenge, we discard the charge-sharing scheme on the same bit-line. Inspired by the DRISA-1T1C, we add the highly optimized 3T XNOR logic circuit after the sense amplifier. Thus, the DRAM cells' operation is no different from the normal read.
- **Data partitioning (Challenge-IV):** Due to the large memory space requirement of the short read alignment algorithm and non-localized computation [40], [42], even having solutions to alleviate the hardware issues as mentioned above, data partitioning will remain a challenge that must be overcome. To address this challenge, we propose a customized alignment-in-memory algorithm, discussed in Section IV, and a new correlated data partitioning method in Section V.

III. Aligner-D Design

In this section, we present *Aligner-D* as an efficient accelerator on top of typical DRAM architecture to execute the DNA short read alignment task. The *Aligner-D*'s memory organization is depicted in Fig. 4 based on the DRAM hierarchy. Each memory matrix (mat) has multiple computational memory sub-arrays linked to two shared modules i.e. a Global Row Decoder (GRD) and a Row Buffer (GRB). The *Aligner-D*'s Controller (Ctrl) is designed to manage the sub-arrays for fully data-parallel in-sub-array operation based on the input data. The row space of *Aligner-D*'s sub-array is split into two distinct regions to support the alignment algorithm: 1- Ref rows that store the nucleotides (NTs) from the reference genome sequence. 2- Marker rows that store the marker table generated by the BWT. The details of such mapping will be introduced in Section V. As shown in Fig. 4(E), the *Aligner-D*'s sub-array is motivated by DRISA [2] where the computing logic gates are following SA, but *Aligner-D* is enhanced and optimized to perform both parallel XNOR-based matching and addition with the novel 3T XNOR gate and the 12T full adder, respectively, as discussed below.

A. Aligner-D Circuits

Our key objective here is to develop the peripheral circuits to execute all the required computations in the alignment-in-memory algorithm that will be introduced in the next section. To achieve this goal, we propose the *Aligner-D* hardware, as shown in Fig. 4, developed on top of the existing DRAM circuitry with a minor area overhead. It mainly consists of

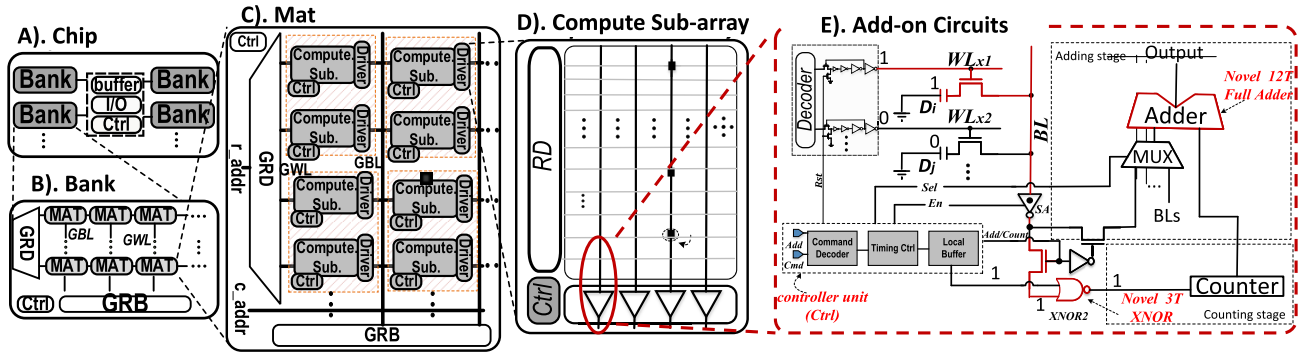
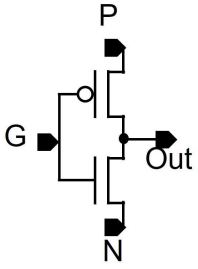


Fig. 4. The *Aligner-D* memory organization and add-on circuits. A). Memory chip architecture. B). Bank structure. C). Mat organization. D). Memory sub-array for computation. E). Add-on circuits for XNOR and ADD operation.

a regular DRAM SA followed by add-on circuits including 3T XNOR and one 3T XNOR based 12T full adder. For the target genome alignment task, both the necessary BW matrix and sampled occurrence table are stored in the same sub-array but at different rows. There are two primary operations during the alignment-in-memory task: `XNOR2` and `addition`. The `XNOR2` is performed among reference nucleotides stored at the local buffer and BW matrix from the sub-array. The `addition` is performed between the `XNOR2` result and marker from sampled occurrence table. Parallel computing comes from multi-bitlines. The rows are activated one by one. When one row is activated, the `XNOR2` is performed at every bitline by our novel 3T XNOR gate, while the `addition` is completed by our 3T XNOR gate based 12T adder.

B. 3T XNOR Logic Circuit

Traditional DRAM-based computing platform has a complex mechanism to realize the bit-wise operation or/and has a potential reliability issue due to the slight sense margin. To overcome these shortages, we proposed the novel `XNOR2` logic circuit with only three transistors for the essential bit-wise operation of the genome alignment task. Inspired by the area and power efficient gate diffusion input (GDI) design [43], where the input may feed through the source or drain side instead of the gate. As shown in Fig. 5, the GDI approach allows the implementation of many complex logic functions with only two transistors. However, some functions may not deliver strong '1' and '0' to the Out terminal. For example, the `AND2` operation needs the source side of PMOS connected to the ground. When A is '0' and B is '1', the Out is driven by the ground at the P terminal through PMOS. Conventionally, we use PMOS to deliver a strong '1' and NMOS to deliver a strong '0'. Using PMOS to deliver '0' or NMOS to deliver '1' will cause the output slightly higher than GND or lower than VDD and lead to weak '1' and '0' signals. Especially, if we combine such GDI gates together for more complex logic, a such weak signal may propagate along the path and result in malfunction. To avoid the above issue and minimize the area/power overhead, we adopt the novel 3T XNOR design to realize the matching operation. Unlike the GDI design where input can come from either PMOS source, NMOS source, or the gate. Our 3T XNOR design, as shown in Fig. 6(A), connects the gate and source of NMOS. The input is associated with the combined port. The source of PMOS is connected with the power supply and the gate is connected with a bias voltage source. When the Out is '0', NMOS delivers the strong



N	P	G	Out
'0'	B	A	$\overline{A}B$
B	'1'	A	A+B
'1'	B	A	A+B
B	'0'	A	AB
C	B	A	$\overline{A}B$+AC
'0'	'1'	A	$\overline{A}$

Fig. 5. Gate diffusion input(GDI) design.

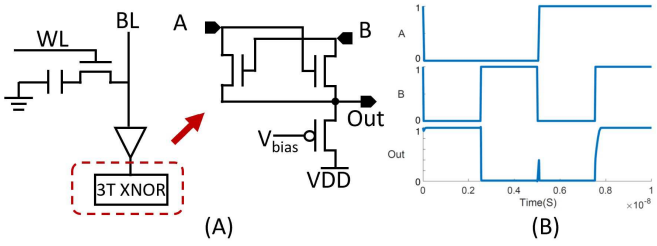


Fig. 6. 3T XNOR.

'0' to the output. When the Out is '1', NMOS cannot provide the strong '1' to output. The bias PMOS hereby compensates the output to the strong '1'. The circuit work condition can be summarized as the following 3 cases:

- Case I: If only one of the inputs, A or B, is '1' and another is '0', there is only one NMOS fully turned on and another NMOS is turned off. The turned-on NMOS will pull down the output to '0'.
- Case II: If both A and B inputs are '1', both NMOS will be turned on. Such NMOS will deliver the weak '1' to the Out terminal. With the help of the biased PMOS, the output is compensated to a strong '1'.
- Case III: If both A and B inputs are '0', both NMOS will be turned off. The output terminal will be driven by the PMOS.

Fig. 6(B) shows the transient simulation waveform. Note that the output delivers strong '1' and '0' in all different combination inputs.

C. 12T Full Adder Circuit

Another important operation in the alignment-in-memory algorithm is the addition. To implement it in Aligner-D, we design a 12T full adder with two aforementioned 3T XNOR gates and one 2-1 MUX. The corresponding Boolean

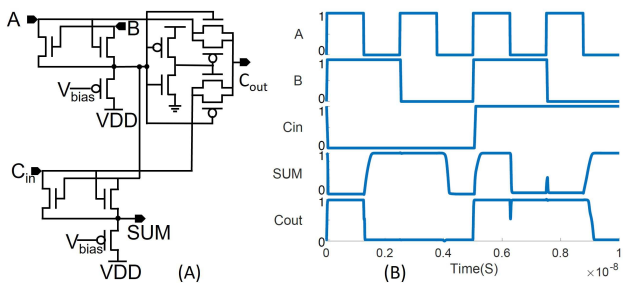


Fig. 7. 12T Adder.

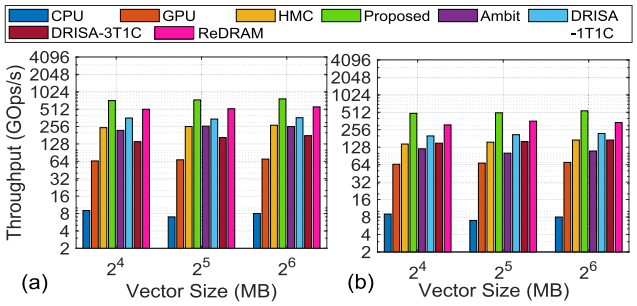


Fig. 8. Throughput results of (a) XNOR2 operation and (b) Addition operation for various computing platforms corresponding to three vector sizes.

equation is given in Eq. 1.

$$SUM = A \odot B \odot C_{in}$$
$$C_{out} = (A(\overline{A \odot B})) + (C_{in}(A \odot B)) \quad (1)$$

Comparing with the traditional 34T full adder design [44], the 12T full adder design saves ∼65% area. Additionally, in the genome alignment task, the matching operation is prior to the addition. Thus the 3T XNOR for matching can be reused as part of the addition circuit to save another ∼10% area. Simulation with NCSU 45nm PDK shows that the proposed 12T full adder only consumes 4.55uW power and the latency is only 0.128ns when running at 800MHz with 1V supply voltage. More importantly, the output is strong '1' and '0'. Thus avoid the reliability issue in those GDI-based full adder designs.

In the alignment task, one of the XNOR2 operands is the reference sequence which needs to be stored in the memory. Another operand is a single nucleotide either 'A', 'C', 'G', or 'T' from the short read. Unlike previous works that store both operands into memory [1], [2], [3]. In this work, we only store the reference sequence in the DRAM memory array. The second operand is stored in the buffer and fed into the 3T XNOR gate which is located after the SA. For the memory array, during the operation, its only need to do the normal read to get the reference genome sequence with the traditional SA. Since there is no multiple-row activation, the charge on BL is not shared with any other rows. The BL voltage is just the same as the commercial DRAM product. Therefore, the reliability of BL sensing is not a concern. Also, the row initialization problem associated with the multiple row activation does not exist in our design. For the normal read operation, the data will be recovered after the sensing stage. We don't need to copy/clone the entire row to a dedicated row for computing. The elimination of row initialization is helpful to reduce the latency and the area overhead.

Finally, the 3T XNOR and 12T full adder design is more energy and area efficient than the multiple row activation designs. For a 128 cols array where marker is stored in 32-bit length integer, assuming other circuit are the same i.e. counters, our proposed design overhead is only 3+(12-3)×32/128 = 5 Transistors/col where the multiple row activation scheme designs require 2(Computing rows) + 4(Extra SA) + 34 × 32/128(Adder) = 14.5 Transistors/col.

D. Performance

1) Throughput: We assess and compare the *Aligner-D*'s throughput with conventional computing platforms such as a Core-i7 Intel CPU [28] and an NVIDIA 1080Ti GPU [29]. There is a great number of reconfigurable or application-specific PIM platforms implemented in or close to the memory die [6], [45], [46]. Due to the lack of space, our comparison is limited to 5 new processing-in-DRAM accelerators, DRISA-1T1C [2], DRISA-3T1C [2], Ambit [1], ReDRAM [3] and HMC 2.0 [27], to run two main bulk bit-wise operations, i.e., XNOR2, and add, required for DNA short read alignment. For an objective comparison, we simulate the platform's throughput with the same memory configuration of 8 banks including 512 × 256 computational sub-arrays. The Intel CPU platform with 4/8 cores/threads works with two 64-bit channels (DDR4-1866/2133). The GPU platform has 3584 CUDA cores working at 1.5GHz [29] and 352-bit GDDR5X. The HMC consists of 32 vaults each with 10 GB/s bandwidth. We then run the two operations repeatedly for different length vectors ($2^{27}/2^{28}/2^{29}$). As shown in Fig. 8, the throughput results of each platform is reported, separately.

Based on our observation 1) the external bandwidth of the main memory restricts the throughput of the Von-Neumann computing platforms, i.e., CPU and GPU. Besides, the internal bandwidth limits the throughput of the HMC platform. In comparison, HMC achieves ∼25× and 6.5× higher throughput compared with the CPU and GPU, respectively, for different bit-wise operations. On the other side, PIM architectures unblock the bottleneck of data transfer and obtain a significantly higher throughput compared with von-Neumann computing systems. We can see that the proposed design achieves on average 71× and 8.2× higher throughput respectively compared with CPU and GPU. 2) For bulk bit-wise XNOR2-based operations, *Aligner-D* easily outperforms other platforms on average by 4×, 2.45×, 3.26×, and 1.65×, respectively, compared with Ambit [1], DRISA-1T1C [2], DRISA-3T1C [2], and ReDRAM [3]. As a result, the *Aligner-D*'s mechanism can be a potential and alternative solution to address Challenge-I by offering a high-throughput bit-wise XNOR-based operation.

2) Energy Consumption: The energy consumption of the processing-in-DRAM platforms and CPU[1] to run three bulk bit-wise operations is estimated per Kilo-Byte. Fig. 9 depicts the energy saving of *Aligner-D* and other platforms normalized to the baseline CPU. For XNOR2 operation, *Aligner-D* reduces the energy consumption by ∼ 23%, 16%, 33.8%, and 22.7%, respectively, compared with Ambit [1], DRISA-1T1C [2], DRISA-3T1C [2], and ReDRAM [3]. We observe that for

[1]The CPU data doesn't consider the processor energy to perform the operation.

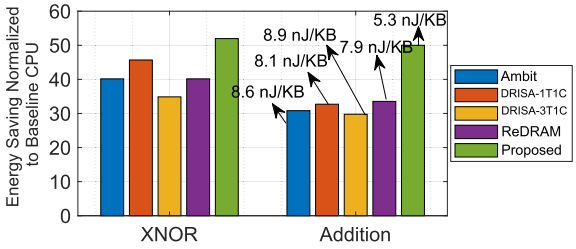


Fig. 9. Energy saving of various in-DRAM computing platforms normalized to CPU baseline.

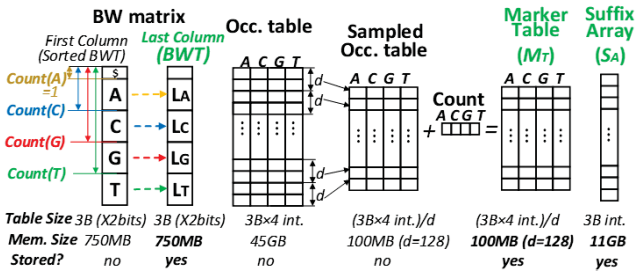


Fig. 10. The required pre-computation in *Aligner-D*'s alignment-in-memory algorithm.

addition operation, the proposed design shows ∼51× higher energy saving over copying data via the DDR4 interface. However, the most energy-efficient in-DRAM platform, i.e., dual-row ReDRAM shows ∼33× energy-efficiency. The energy consumption number per KB for all platforms is also reported for the addition operation.

3) Area: We estimate *Aligner-D* and other PIM platforms' footprint on top of the typical DRAM chip. Despite the standard modules, i.e., the traditional SA, decoder, pre-charge circuit, for a 256 x 128 sized DRAM sub-array, the Ambit requires additional 24 transistors per BL for the DCC rows and 24 transistors per BL for the second SA, that equivalent to ∼ 18.75% of the chip area. For each BL, the DRISA-3T1C requires another $256 \times 2 = 512$ transistors for the 3T DRAM cells and 12 transistors for the second SA like Ambit. The overall area overhead is ∼ 2.05× of the traditional DRAM die. The DRISA-1T1C demands eight transistors for the XNOR gate, 16 transistors for a latch, and 12 transistors for the second SA on every BL. The total area overhead is ∼ 14.06%. The ReDRAM needs 22 transistors per BL for modified SAs, and two more transistors in the decoder buffer chain, equivalent to ∼ 9.3% of the chip area. In contrast, as we analyzed above, our *Aligner-D* only requires five transistors per BL. Only ∼ 1.95% area overhead makes it the most area-efficient design.

IV. Alignment-in-Memory Algorithm

The DNA alignment-in-memory algorithm consists of two stages: exact alignment and inexact alignment [6], [41], [47]. For most sequencing data, up to ∼70% of short reads should be exactly aligned to the reference genome after stage one. The remaining reads are then processed through stage two. Most genome variations are relatively small, involving only one or two nucleotides. If we only allow an exact match between short reads and the reference genome, the reads containing the genome variations from the sample cannot map to the reference genome. In addition, the genome variations (e.g., single nucleotide mutations) cannot be identified based on the exact alignment algorithm. Thus, such potential molecular signatures cannot be applied for disease phenotype prediction. In the following, we elaborate on these two stages respectively.

A. Exact Alignment Algorithm

The alignment algorithm is developed based on BWT and FM-Index [8], and optimized using *Aligner-D*'s functions. As depicted in Fig. 10, the first step is to store some important pre-computed tables based on reference genome S. This is only a single-time computation for BWT, S_A, and Marker Table (M_T) to be saved in the *Aligner-D* consuming ∼12GB of memory space. We need to reconstruct and store the table data into different memory sub-arrays, banks, and chips to provide high-speed memory access and parallel PIM operations, as discussed in Section V. In Fig. 10, the number of nucleotides in the BW matrix's first column that are lexicographically smaller than the nucleotide-nt is represented by $Count(nt)$. So there are four elements for sequence alignment. The FM-index table so-called Occurrence (Occ.) table, is then made based on BWT. In Occ. table, each element-$Occ[i, nt]$ represents the number of occurrences in the position range 0 to $i-1$ for nucleotide-nt in the BWT. We sampled such large Occ. table every d position (i.e., bucket width) and reconstructed a Sampled Occ. table. Therefore, we are able to diminish the table size by a factor of d. Besides, we developed a M_T by element-wise addition between $Count(nt)$ and sampled Occ. table. Such a marker table has the same size as Sampled Occ-Table. M_T basically consists of the matched position of nt in the first column of BWT. Accordingly, *Aligner-D* is able to effectively retrieve low and $high$ values in each iteration.

In Algorithm 1, the backward search process can be reconstructed with the presented hardware-friendly $Bound(M_T, nt, id)$ procedure (line-12) executed on BWT. This procedure calculates the updated low or $high$ interval's value from M_T with an input index-id considering a bucket width of d. As can be observed, such $Bound$ procedure iteratively performs in every step of the 'for' loop. The *Aligner-D* is particularly developed to run such intensive computation via computing two operations i.e. matching and addition between the occurrence counting data and 'marker' value for the required nts located between checkpoint positions and remaining positions in BWT. As indicated in Algorithm 1, we leverage 3 *Aligner-D*'s in-memory functions, named, MEM (memory read operation), $XNOR\_Match$ (XNOR2), and IM_ADD (add) to implement the $Bound$ procedure completely within memory. MEM function is used to access data in the stored S_A or M_T having the index. $XNOR\_Match$ conducts an in-memory XNOR2 operation to check if there is a match between the BWT elements saved in the entire word-line and the current input-nt in a single computational cycle. IM_ADD conducts 32-bit integer in-memory addition operation (index range) and computes 'marker+count_match' results without sending it to CPU or other computing units. We identify two main features in the modified alignment algorithm that make it a potential candidate for in-memory implementation: 1) it matches *Aligner-D*'s logic operations (e.g. matching and addition) very well and 2) it is memory-bound and parallelizable, not needing any memory access to perform entire read alignment.

Algorithm 1 DNA Exact Alignment-in-Memory

Require: : Pre-Compute and Data Mapping to *Aligner-D*: Partition pre-computed BWT, Marker Table (M_T) and Suffix Array (S_A) into *Aligner-D* chip.
input: DNA Short Read-R
output: positions of short read-R in reference genome-S
Step-1. Initialization:
1: $low \leftarrow 0$, $high \leftarrow |S| - 1$
Step-2. Backward Search:
2: **for** $i := |R| - 1$ to 0 **do**
3: $low \leftarrow$ ***Bound***$(M_T[\lfloor low/d \rfloor], R[i], low)$
4: $high \leftarrow$ ***Bound***$(M_T[\lfloor high/d \rfloor], R[i], high)$
5: **if** $low \geq high$ **then**
6: **break & return 0** ▷ there is no exact alignment
7: **end if**
8: **end for**
Step-3. Get matched positions from stored suffix array based on a search result:
9: **for** $j := low$ to $high - 1$ **do**
10: $positions \leftarrow$ **MEM**$(S_A[j])$ ▷ Read positions from Suffix Array memory
11: **end for**
Define procedure Bound:
12: **Procedure:** ***Bound***(M_T, nt, id) ▷ compute matched interval
13: $count\_match \leftarrow 0$
14: **for** $j := 0$ to $j < (id \bmod d)$ **do** ▷ count number of nt within the BWT region
15: **if XNOR_Match**$(nt, BWT[id - (id \bmod d) + j]) == 1$ **then**
16: $count\_match = count\_match + 1$
17: **end if**
18: **end for**
19: $marker \leftarrow$ **MEM**$(M_T[\lfloor id/d \rfloor], nt])$ ▷ Read Marker Table value
20: **return IM-ADD**$(marker, count\_match)$
21: **end Procedure**

Algorithm 2 DNA Inexact Alignment-in-Memory

Require: : Pre-Compute and Data Mapping to *Aligner-D*: Partition pre-computed BWT, Marker Table (M_T) and Suffix Array (S_A) into *Aligner-D* chip.
input: DNA Short Read-R, z mismatches allowed in the alignment.
output: positions of short read-R in reference genome-S with up to z mismatches.
Step-1. Initialization:
1: $low \leftarrow 0$, $high \leftarrow |S| - 1$
2: **return I= InexactRecursive**$(R, |R|, low, high, z)$:
3: **for** $i := |I| - 1$ to 0 **do**
4: $positions \leftarrow$ **MEM**$(S_A[I[i]])$
5: **end for**
Define procedure InexactRecursive :
6: **Procedure:** ***InexactRecursive***$(R, i, low, high, z)$ ▷ z is the number of mismatches allowed
7: **if** $z < 0$ **then**
8: **break & return 0**
9: **end if**
10: **if** $i < 0$ **then**
11: **return [low,high]**
12: **end if**
13: $I \leftarrow 0$
14: $I \leftarrow I \cup InexactRecursive(R, i - 1, low, high, z - 1)$ ▷ Insertion
15: **for** each $b \in \{A, C, G, T\}$ **do**
16: $low \leftarrow$ ***Bound***$(M_T[\lfloor low/d \rfloor], R[i], low)$
17: $high \leftarrow$ ***Bound***$(M_T[\lfloor high/d \rfloor], R[i], high)$
18: **if** $low < high$ **then**
19: $I \leftarrow I \cup InexactRecursive(R, i, low, high, z - 1)$ ▷ Deletion
20: **if** $b = R[i]$ **then**
21: $I \cup InexactRecursive(R, i - 1, low, high, z)$ ▷ Exact Match
22: **else**
23: $I \cup InexactRecursive(R, i - 1, low, high, z - 1)$ ▷ Inexact Match
24: **end if**
25: **end if**
26: **end for**
27: **return** I
28: **end Procedure**

Throughout the alignment process, the XNOR2 operation mainly involves two vectors: the reference genome and the short read from the patient's DNA. The size of both vectors may vary. The vector size affects the computation and storage efficiency but does not affect the computation accuracy. As shown in Fig. 10, the larger interval d helps to reduce the table size for the storage, but it also requires up to d times XNOR operation for every input nucleotide which may slow down the performance and cause extra energy consumption. For the short read, the size of the vector determines how many nucleotides we need to process by repeating the XNOR2 and add operations. Therefore, the size of the vector does not affect the accuracy of either add or XNOR2 operation but the computation/energy efficiency.

B. Inexact Match

Here we extend the exact alignment algorithm to handle inexact match (mismatch, insertion, and deletion) as shown in Algorithm-2. With recursively computing the intervals that match $R[0, i]$, the presented inexact alignment algorithm allows mismatches between read R and reference genome S within a tolerance (no more than z differences) with the condition that $R[i+1]$ matches $\{low, high\}$. While updating the intervals I, we consider all possible alignments as long as there exists tolerance for differences up to the current position i. For the intervals I of position i, we perform union for all match (line 21) and mismatch (line 23) intervals. Accordingly, the algorithm reports the target positions (line 4) in the reference genome, with no more than z mismatches, to which the read can be mapped to. We observe that since Algorithm-2 again iteratively exploits the presented *Bound* function, it can be also accelerated by *Aligner-D* platform.

V. Aligner-D Hardware Mapping

A. Correlated Data Partitioning

Since the alignment-in-memory algorithm requires a large memory space to store pre-computed tables namely BWT and marker table(M_T), we partition these tables to fully employ *Aligner-D*'s parallelism and to maximize the throughput of alignment computation. The memory region accessed for M_T and BWT, within a BWT index range, can be readily anticipated and computation can be localized by using one common memory sub-array to store such correlated data. Therefore, a new data partitioning method and mapping technique is presented, as depicted in Fig. 11. Such partition technique first saves the correlated regions of BWT and M_T vectors locally in a similar sub-array and then makes local computation possible. As a result, without inter-bank/chip data transfer, $XNOR\_Match$ and IM_ADD operations can be completely implemented within the same memory sub-array. This can potentially address Challenge-IV discussed in Section II-C.

Aligner-D platform has multiple memory chips, each consisting of memory banks, mats, and sub-arrays in a hierarchical fashion. We divide each sub-array (with the size of 256 rows×128 columns) into two particular regions to save two various data types, BWT, and M_T. We assign the first 128 rows to the corresponding BWT. Each row contains up to 64 bps, encoded by two bits. Such BWT rows are mainly developed to perform the parallel matching operation based on $XNOR\_Match$. We store the marker's values M_T next to the BWT region. The M_T is pre-computed with checkpointed d positions (=64) for every row. To hold the size in check for *Aligner-D*, we save M_T horizontally in 128 rows, each row storing 128-bit (4-byte value for bps). We use identical colors in Fig. 11 to indicate the BWT region and the corresponding M_T region. After data partitioning and mapping the data, *Aligner-D*'s ctrl starts from the rightmost symbol in R and takes two steps to carry out the *Bound* function and return $high$ and low for the next symbol as detailed below.

B. Parallel Search

Given an input nucleotide A and input index as id in Fig. 11, the *Aligner-D*'s controller easily transforms the BWT index to the corresponding addresses for WL and BL saving data $BWT[id - (id \bmod d)]$ to $BWT[id]$. These bits are read out simultaneously by the SAs and given to

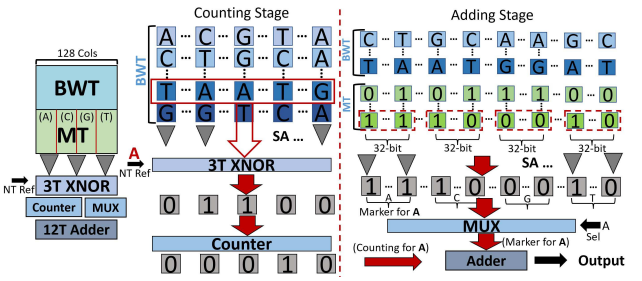


Fig. 11. The memory sub-array partitioning in *Aligner-D* platform to perform Parallel search and Rank computation.

the 3T XNOR part along with the input nucleotide A for the parallel search operation named $XNOR\_Match$. When XNOR2 output equals '1', meaning a match is detected, then $count\_match$ operation needs to be performed. This is accomplished through the embedded counter in the ctrl unit that counts up. The $XNOR\_Match$ function is shown in Fig. 11 counting stage to locate As in a single memory sub-array. By finishing the counting, the memory sub-array the $count\_match$ and $marker\_add$ (marker address) are returned. It is noteworthy that such a correlated partitioning technique offers local memory access for the read operation (MEM) of $marker$ in the same memory array.

C. Parallel Rank Calculation

The *Aligner-D* buffers the $count\_match$ and $marker$, respectively, in the counter and M_T regions, as depicted in Fig. 11, to carry out IM_ADD function. To maximize the throughput, with n active sub-arrays, each sub-array can readily compute the parallel add of 32-bit. Fig. 11 adding stage delineates the organization of sub-array for such parallel operation. In total, 32 aforementioned 12T adders are chained up to build a carry ripple adder(CRA). The operands of the CRA are from the counter and Marker table stored in the M_T region respectively. To handle the add operation, as shown in the example provided in Fig. 11 adding a stage, a MUX is placed on the top of CRA. In the example, four 32-bit data are read from the M_T region. The MUX chooses one of the four 32-bit data based on the input nucleotide and feeds it to the 32-bit CRA. Another operand of CRA is coming from the previous counting stage as the counting result. The bit-by-bit addition begins with the LSBs of two operands and goes on towards MSBs.

VI. Experiments on Alignment Task

A. Setup

1) Evaluation Framework: We evaluate the performance of *Aligner-D* as a novel PIM architecture with an extensive cross-layer framework by developing two in-house simulators. 1- At the circuit level, *Aligner-D*'s memory sub-array and the peripheral circuits (3T XNOR, SA, etc.) were first developed using Cadence Spectre with 45nm technology process of NCSU product development kit library [48]. We used this to evaluate the design and obtain different performance parameters. We then used Verilog-HDL and the Design Compiler [49] with a 45nm industry library to assess the memory ctrl circuits. 2- An architecture-level simulator is designed based on NVmain [50], as a cycle-accurate main memory simulator.

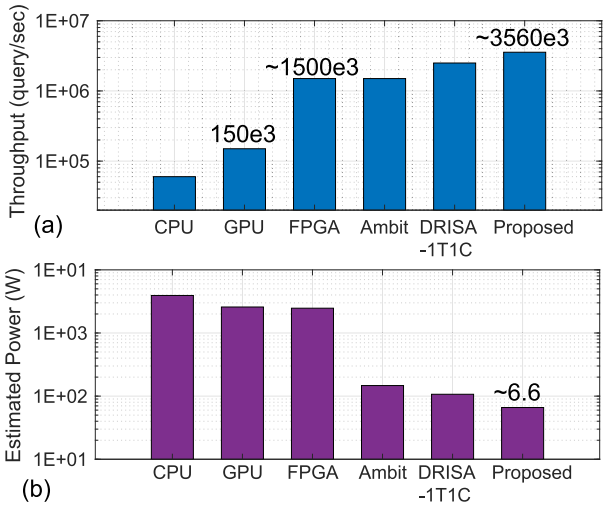


Fig. 12. (a) Log scaled power consumption and (b) throughput of different accelerators compared to *Aligner-D*.

We feed the circuit-level performance data into the developed simulator. It is designed to alter the configuration files with respect to different memory organizations and outputs performance results for *Aligner-D*'s PIM operations. 3- We then used Matlab to develop a behavioral-level simulator. The simulator takes the architecture results to compute the energy and latency parameters for *Aligner-D*. To increase the performance corresponding to the available resources, we also integrated a mapping optimization technique into the platform.

2) Different Accelerators: Recently processing-in-memory-based accelerator designs have been proposed to accelerate the DNA alignment task. Such as, PARC [51] and BioSEAL [52] presented CAM-based accelerator designs. PIM-Align [53] proposed an accelerator for standard FM-Index search with 3D-stacked DRAM technology. Reference [54] proposed PIM accelerator for the BWT-based short read alignment with spintronic computational RAM (MRAM). Considering the versatility and the mass of data in the alignment task, we conduct a comprehensive comparison with ReCAM [39], a dynamic programming based acceleration solution along with FM-Index acceleration solutions including: Soap2 [9]/Soap3-dp [9] on CPU/GPU, FPGA [12], and two counterpart processing-in-DRAM accelerators, i.e., Ambit [1] and DRISA-1T1C [2].

For the sake of space, we refer the readership to the aforementioned works for more details on each accelerator's implementation. For short read alignment task on CPU/GPU with reads of $\leq$2 mismatches, we used Soap2/Soap3 [9]. For solid comparison amongst *Aligner-D* and other computing platforms, the human genome Hg19[2] was taken into account. Accordingly, 10 million 100-bp read queries were generated via ART simulator [55] and aligned to the human genome to report the performance.

B. Results

1) Power-Throughput-Area Trade-Offs: Figure 12a and b respectively show the throughput and power consumption of different computing platforms. *Aligner-D* shows the highest throughput (3560K query/sec) compared to CPU [9], GPU [9], FPGA [12], Ambit [1] and DRISA-1T1C [2]

[2]Population Variation = 0.1% and Genome Error Rate = 0.2%.

TABLE I
PERFORMANCE OF READ ALIGNMENT ACCELERATORS

	FM-Index						Dynamic programming
metrics	CPU	GPU	FPGA	Ambit	DRISA	Aligner-D	ReCAM
Throughput/Watt	153	581.3	6.1K	102K	230K	539K	26.81
Throughput/Watt/mm^2	0.011	0.39	0.42	28.66	65.63	219.26	0.24

platforms due to its massively-parallel and local computational scheme. We observe that Ambit and the FPGA implementations achieve almost the same throughput as a result of the multi-cycle in-DRAM XOR operation. However, DRISA-1T1C stands as the second high-performance platform. Besides, thanks to the simple structure that *Aligner-D* needs less transistor per column than DRISA-1T1C. We observe that *Aligner-D* consumes significantly lower power (∼6.6W) along with other PIM designs compared to other von-Neumann computing platforms. Taking the chip area into consideration, the existing trade-offs among power, throughput, and area for different accelerators can be understood by correlated parameters, as tabulated in Table I. A 16GB *Aligner-D* required for DNA short read alignment takes ∼4160 mm^2 area, where the GPU [9] and CPU [9] occupies 14300 and 1600 mm^2, respectively. Note that, based on our experiment, Ambit [1] imposes the least area-overhead by ∼3200mm^2 among FM-Index based methods.

Based on Table I, we observe that *Aligner-D* outperforms different accelerators w.r.t. throughput/Watt. *Aligner-D* can improve the read alignment's throughput per Watt by ∼20104×, ∼3522×, ∼927×, ∼88×, ∼5.28×, and ∼2.34×, over ReCAM [39], CPU, GPU, FPGA, Ambit, and DRISA-1T1C, respectively. Moreover, Table I reports throughput per Watt per mm^2 for different accelerators. Considering the area parameter, we see that *Aligner-D* can remarkably boost the alignment performance compared with all the other designs. *Aligner-D* achieves ∼3.34× higher throughput per Watt per mm^2 compared to the DRISA-1T1C. To sum it up, *Aligner-D* offers a parallel processing-in-memory scheme and ultra-high internal bandwidth features that can accelerate short read alignment task. It is noteworthy that Aligner-D is more hardware-friendly. Since we did not directly modify the standard DRAM subarray design and the add-on circuit only needs a few transistors per column. Aligner-D incurs less than ∼2% area overhead on top of the typical DRAM chip. Such a small area overhead doesn't necessarily force DRAM manufacturers to reduce the DRAM capacity to maintain the current dimensions in the DIMM form factors. To have a quantitative comparison, in our evaluation, a 32 Mbit single-bank DRAM with 512-bit data width shows 4.53 mm^2 area and 321 mW leakage power, while Aligner-D imposes 4.62 mm^2 (∼ 2% ↑) area and 334 mW (∼ 4% ↑) leakage power.

2) Memory Wall: The off-chip memory access needed for different computing platforms is reported in Fig. 13a. Based on our evaluation, all the FM-Index-based platforms, i.e., CPU/GPU [9] and FPGA [12], excluding in-memory accelerators, i.e., Ambit, DRISA-1T1C, and *Aligner-D* are highly dependent on off-chip memory access. It means such platforms consume a huge amount of energy just for fetching data queries and tables in the memory.

The Memory Bottleneck Ratio (MBR) is reported in Fig. 13b. We define MBR as the time fraction needed for data transfer from/to on-chip or off-chip, when the computation has

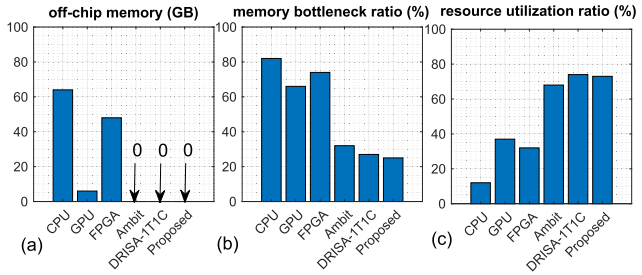


Fig. 13. (a) The off-chip memory access, (b) Memory Bottleneck Ratio, and (c) Resource Utilization Ratio of different computing platforms.

to wait for data, i.e., memory wall happens. The peak throughput for each design is taken into account for performing the assessment. This evaluation mainly considers the number of memory access. We can see that PIM-based *Aligner-D*, Ambit, and DRISA-1T1C consume less than ∼35% time for data transfer and memory access. The Resource Utilization Ratio (RUR) is also reported in Fig. 13c. The less MBR can be understood as a higher RUR. We can see that with up to ∼75%, DRISA-1T1C and *Aligner-D* achieve the highest RUR. Taking everything into account, PIM acceleration schemes offer a high ratio (>60%) confirming the conclusion drawn in Fig. 13b. The memory wall evaluation shows the efficiency of the *Aligner-D* platform for solving the memory wall challenge.

VII. POTENTIAL CHALLENGE & APPLICATION

There are still many unsolved issues for using processing-in-memory technology to accelerate Next-Generation Sequencing (NGS). We leave these possible challenges as the future works. e.g., *Aligner-D* is highly optimized for the BWT-based DNA short-read alignment task. Thus, it is not suitable for long-read alignment or genome assembly tasks. Those are also data-intensive processes in genome sequencing. Since the *Aligner-D* is a DRAM-based accelerator, the notorious "Row Hammer" [56] attack may still distort the reference genome data stored in the DRAM array. The reference genome data is pre-processed and used for every input nucleotide. The attack will lead to the malfunction of all the short-read inputs which feed after the data distortion.

On the other hand, although *Aligner-D* was proposed for the specific alignment task, other applications could easily leverage this architecture as well. Since the *Aligner-D* could efficiently perform the parallel `XNOR2` operation where one of the operands is from the outside of the array. With the help of counter and adder, *Aligner-D* could achieve the multiplication and addition (MAC) operation in the purely digital domain. This is a natural fit for the popular Deep Neural Network application [57] where weights are stored inside the DRAM array and the activations are fed from buffer/IO. Other popular algorithms such as Advanced Encryption Standard (AES) [58] and Min/Max Searching [59] algorithms can also benefit from the *Aligner-D* by the high-parallel and efficient `XNOR2` operation.

VIII. CONCLUSION

In this work, we presented *Aligner-D*, as a high-throughput and energy-efficient PIM architecture to address some of the existing issues in state-of-the-art DRAM-based acceleration

solutions for performing bulk bit-wise XNOR-based operations i.e. limited throughput, row initialization, reliability concerns, etc. incurring less than 2% area overhead on top of the typical DRAM chip. Here, we design a highly parallel and customized read alignment algorithm for *Aligner-D* that only requires the presented in-memory logic operations. To accelerate and support both *exact* and *inexact* match tasks, *Aligner-D* is then configured with a novel data partitioning and mapping technique that provides local storage and processing of DNA sequence to fully utilize the algorithm-level's parallelism. *Aligner-D* improves the short read alignment throughput per Watt by ∼20104×, ∼3522×, ∼927×, ∼88×, ∼5.28×, and ∼2.34×, over ReCAM, CPU, GPU, FPGA, Ambit, and DRISA-1T1C, respectively. To the best of our knowledge, this is the first work that proposes an XNOR-friendly processing-in-DRAM accelerator that can be applied to a variety of applications including DNA alignment, assembly, data encryption, etc.

REFERENCES

[1] V. Seshadri et al., "Ambit: In-memory accelerator for bulk bitwise operations using commodity DRAM technology," in *Proc. 50th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Oct. 2017, pp. 273–287.

[2] S. Li, D. Niu, K. T. Malladi, H. Zheng, B. Brennan, and Y. Xie, "DRISA: A DRAM-based reconfigurable in-situ accelerator," in *Proc. 50th Annu. IEEE/ACM Int. Symp. Microarchit.*, Oct. 2017, pp. 288–301.

[3] S. Angizi and D. Fan, "ReDRAM: A reconfigurable processing-in-DRAM platform for accelerating bulk bit-wise operations," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2019, pp. 1–8.

[4] Y. Turakhia, G. Bejerano, and W. J. Dally, "Darwin: A genomics co-processor provides up to 15,000× acceleration on long read assembly," in *Proc. 23rd Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2018, pp. 199–213.

[5] F. Zokaee, H. R. Zarandi, and L. Jiang, "AligneR: A process-in-memory architecture for short read alignment in ReRAMs," *IEEE Comput. Archit. Lett.*, vol. 17, no. 2, pp. 237–240, Jul. 2018.

[6] S. Angizi, J. Sun, W. Zhang, and D. Fan, "AlignS: A processing-in-memory accelerator for DNA short read alignment leveraging SOT-MRAM," in *Proc. 56th Annu. Design Autom. Conf.*, Jun. 2019, pp. 1–6.

[7] B. Langmead, C. Trapnell, M. Pop, and S. L. Salzberg, "Ultrafast and memory-efficient alignment of short DNA sequences to the human genome," *Genome Biol.*, vol. 10, no. 3, p. R25, 2009.

[8] H. Li and R. Durbin, "Fast and accurate short read alignment with Burrows–Wheeler transform," *Bioinformatics*, vol. 25, no. 14, pp. 1754–1760, 2009.

[9] R. Luo et al., "SOAP3-dp: Fast, accurate and sensitive GPU-based short read aligner," *PLoS ONE*, vol. 8, no. 5, May 2013, Art. no. e65632.

[10] A. Madhavan, T. Sherwood, and D. Strukov, "Race logic: A hardware acceleration for dynamic programming algorithms," in *Proc. ACM/IEEE 41st Int. Symp. Comput. Archit. (ISCA)*, Jun. 2014, pp. 517–528.

[11] Y.-C. Wu, C.-H. Chang, J.-H. Hung, and C.-H. Yang, "A 135-mW fully integrated data processor for next-generation sequencing," *IEEE Trans. Biomed. Circuits Syst.*, vol. 11, no. 6, pp. 1216–1225, Dec. 2017.

[12] J. Arram, T. Kaplan, W. Luk, and P. Jiang, "Leveraging FPGAs for accelerating short read alignment," *IEEE/ACM Trans. Comput. Biol. Bioinf.*, vol. 14, no. 3, pp. 668–677, May 2017.

[13] S. Mittal, "A survey of ReRAM-based architectures for processing-in-memory and neural networks," *Mach. Learn. Knowl. Extraction*, vol. 1, no. 1, pp. 75–114, 2018.

[14] P. Chi et al., "Prime: A novel processing-in-memory architecture for neural network computation in ReRAM-based main memory," *ACM SIGARCH Comput. Archit. News*, vol. 44, no. 3, pp. 27–39, 2016.

[15] A. Shafiee et al., "ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," *ACM SIGARCH Comput. Archit. News*, vol. 44, no. 3, pp. 14–26, 2016.

[16] A. D. Patil, H. Hua, S. Gonugondla, M. Kang, and N. R. Shanbhag, "An MRAM-based deep in-memory architecture for deep neural networks," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2019, pp. 1–5.

[17] B. Chen, F. Cai, J. Zhou, W. Ma, P. Sheridan, and W. D. Lu, "Efficient in-memory computing architecture based on crossbar arrays," in *IEDM Tech. Dig.*, Dec. 2015, p. 17.

[18] F. Zokaee, M. Zhang, and L. Jiang, "FindeR: Accelerating FM-index-based exact pattern matching in genomic sequences through ReRAM technology," in *Proc. 28th Int. Conf. Parallel Archit. Compilation Techn. (PACT)*, Sep. 2019, pp. 284–295.

[19] C. Nail et al., "Understanding RRAM endurance, retention and window margin trade-off using experimental results and simulations," in *IEDM Tech. Dig.*, Dec. 2016, pp. 4–5.

[20] M. Hu et al., "Dot-product engine for neuromorphic computing: Programming 1T1M crossbar to accelerate matrix-vector multiplication," in *Proc. 53rd Annu. Design Autom. Conf.*, Jun. 2016, pp. 1–6.

[21] S. Aga, S. Jeloka, A. Subramaniyan, S. Narayanasamy, D. Blaauw, and R. Das, "Compute caches," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2017, pp. 481–492.

[22] C. Eckert et al., "Neural cache: Bit-serial in-cache acceleration of deep neural networks," 2018, *arXiv:1805.03718*.

[23] G. Dai et al., "GraphH: A processing-in-memory architecture for large-scale graph processing," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 4, pp. 640–653, Mar. 2018.

[24] S. Angizi and D. Fan, "GraphiDe: A graph processing accelerator leveraging in-DRAM-computing," in *Proc. Great Lakes Symp. VLSI*, May 2019, pp. 45–50.

[25] R. Zhou, S. Tabrizchi, A. Roohi, and S. Angizi, "LT-PIM: An LUT-based processing-in-DRAM architecture with RowHammer self-tracking," *IEEE Comput. Archit. Lett.*, vol. 21, no. 2, pp. 141–144, Jul. 2022.

[26] R. Zhou, A. Roohi, D. Misra, and S. Angizi, "ReD-LUT: Reconfigurable in-DRAM LUTs enabling massive parallel computation," in *Proc. 41st IEEE/ACM Int. Conf. Comput.-Aided Design*, Oct. 2022, pp. 1–8.

[27] *Hybrid Memory Cube Specification 2.0*, Micron Technol., Lehi, UT, USA, 2018. [Online]. Available: https://www.micron.com/-/media/client/global/documents/products/data-sheet/hmc/gen2/hmc_gen2.pdf

[28] *6th Generation Intel Core Processor Family Datasheet*, Intel, Santa Clara, CA, USA, 2022. [Online]. Available: https://cdrdv2.intel.com/v1/dl/getContent/332687

[29] *Geforce GTX 1080 TI*, Nvidia, Santa Clara, CA, USA, 2022. [Online]. Available: https://www.nvidia.com/en-us/geforce/products/10series/geforce-gtx-1080-ti./

[30] Y. Kim, W. Yan, and O. Mutlu, "Ramulator: A fast and extensible DRAM simulator," *IEEE Comput. Archit. Lett.* vol. 15, no. 1, pp. 45–49, Jan. 2016.

[31] S. Angizi and D. Fan, "Accelerating bulk bit-wise X(N)OR operation in processing-in-DRAM platform," 2019, *arXiv:1904.05782*.

[32] V. Seshadri et al., "Fast bulk bitwise and and OR in DRAM," *IEEE Comput. Archit. Lett.*, vol. 14, no. 2, pp. 127–131, Jul./Dec. 2015.

[33] V. Seshadri et al., "RowClone: Fast and energy-efficient in-DRAM bulk data copy and initialization," in *Proc. 46th Annu. IEEE/ACM Int. Symp. Microarchit.*, Dec. 2013, pp. 185–197.

[34] M. F. Ali, A. Jaiswal, and K. Roy, "In-memory low-cost bit-serial addition using commodity DRAM technology," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 67, no. 1, pp. 155–165, Jan. 2020.

[35] G. Sideris, "Intel 1103-MOS memory that defied cores," *Electronics*, vol. 46, no. 9, pp. 108–113, 1973.

[36] Q. Deng, L. Jiang, Y. Zhang, M. Zhang, and J. Yang, "DrAcc: A DRAM based accelerator for accurate CNN inference," in *Proc. 55th ACM/ESDA/IEEE Design Autom. Conf. (DAC)*, Jun. 2018, p. 168.

[37] S. Canzar and S. L. Salzberg, "Short read mapping: An algorithmic tour," *Proc. IEEE*, vol. 105, no. 3, pp. 436–458, Mar. 2017.

[38] W. Huangfu, S. Li, X. Hu, and Y. Xie, "RADAR: A 3D-ReRAM based DNA alignment accelerator architecture," in *Proc. 55th ACM/ESDA/IEEE Design Autom. Conf. (DAC)*, Jun. 2018, p. 59.

[39] R. Kaplan, L. Yavits, R. Ginosar, and U. Weiser, "A resistive CAM processing-in-storage architecture for DNA sequence alignment," *IEEE Micro*, vol. 37, no. 4, pp. 20–28, Aug. 2017.

[40] J. S. Kim et al., "GRIM-filter: Fast seed location filtering in DNA read mapping using processing-in-memory technologies," *BMC Genomics*, vol. 19, no. 2, p. 89, 2018.

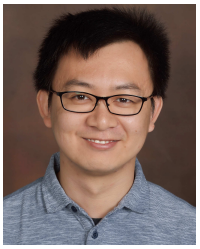
[41] S. Angizi, W. Zhang, and D. Fan, "Exploring DNA alignment-in-memory leveraging emerging SOT-MRAM," in *Proc. Great Lakes Symp. VLSI*. New York, NY, USA: Association for Computing Machinery, Sep. 2020, pp. 277–282.

[42] R. Wertenbroek and Y. Thoma, "K-mer counting with FPGAs and HMC in-memory operations," in *Proc. NASA/ESA Conf. Adapt. Hardw. Syst. (AHS)*, Aug. 2018, pp. 233–240.
[43] A. Morgenshtein, A. Fish, and I. A. Wagner, "Gate-diffusion input (GDI): A power-efficient method for digital combinatorial circuits," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 10, no. 5, pp. 566–581, Oct. 2002.
[44] V. Nafeez, M. V. Nikitha, and M. P. Sunil, "A novel ultra-low power and PDP 8T full adder design using bias voltage," in *Proc. 2nd Int. Conf. Converg. Technol. (I2CT)*, Apr. 2017, pp. 1069–1073.
[45] A. Akerib and E. Ehrman, "Non-volatile in-memory computing device," U.S. Patent 14 588 419, May 14, 2015.
[46] S. Angizi, Z. He, and D. Fan, "PIMA-logic: A novel processing-in-memory architecture for highly flexible and energy-efficient logic computation," in *Proc. 55th ACM/ESDA/IEEE Design Autom. Conf. (DAC)*, Jun. 2018, pp. 1–6.
[47] S. Angizi, J. Sun, W. Zhang, and D. Fan, "PIM-Aligner: A processing-in-MRAM platform for biological sequence alignment," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2020, pp. 1265–1270.
[48] *Ncsu EDA freepdk45*, North Carolina State Univ., Raleigh, NC, USA, 2011. [Online]. Available: https://eda.ncsu.edu/freepdk/freepdk45/
[49] *Synopsys Design Compiler*, Synopsys, Mountain View, CA, USA, Sep. 2014.
[50] M. Poremba, T. Zhang, and Y. Xie, "NVMain 2.0: A user-friendly memory simulator to model (non-)volatile memory systems," *IEEE Comput. Archit. Lett.*, vol. 14, no. 2, pp. 140–143, Jul. 2015.
[51] F. Chen, L. Song, H. Li, and Y. Chen, "PARC: A processing-in-CAM architecture for genomic long read pairwise alignment using ReRAM," in *Proc. 25th Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2020, pp. 175–180.
[52] R. Kaplan, L. Yavits, and R. Ginosasr, "BioSEAL: In-memory biological sequence alignment accelerator for large-scale genomic data," in *Proc. 13th ACM Int. Syst. Storage Conf.* New York, NY, USA: Association for Computing Machinery, May 2020, pp. 36–48.
[53] X.-Q. Li, G.-M. Tan, and N.-H. Sun, "PIM-Align: A processing-in-memory architecture for FM-index search algorithm," *J. Comput. Sci. Technol.*, vol. 36, no. 1, pp. 56–70, Jan. 2021.
[54] Z. I. Chowdhury et al., "A DNA read alignment accelerator based on computational RAM," *IEEE J. Explor. Solid-State Comput. Devices Circuits*, vol. 6, pp. 80–88, 2020.
[55] W. Huang, L. Li, J. R. Myers, and G. T. Marth, "ART: A next-generation sequencing read simulator," *Bioinformatics*, vol. 28, no. 4, pp. 593–594, 2012.
[56] O. Mutlu and J. S. Kim, "RowHammer: A retrospective," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 8, pp. 1555–1571, May 2020.
[57] S. Angizi et al., "Accelerating deep neural networks in processing-in-memory platforms: Analog or digital approach?" in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Jul. 2019, pp. 197–202.
[58] Z. He, S. Angizi, F. Parveen, and D. Fan, "Leveraging dual-mode magnetic crossbar for ultra-low energy in-memory data encryption," in *Proc. Great Lakes Symp. VLSI*, New York, NY, USA: Association for Computing Machinery, May 2017, pp. 83–88.
[59] F. Zhang, S. Angizi, and D. Fan, "Max-PIM: Fast and efficient max/min searching in DRAM," in *Proc. 58th ACM/IEEE Design Autom. Conf. (DAC)*, Dec. 2021, pp. 211–216.

Fan Zhang (Graduate Student Member, IEEE) received the M.S. degree in electrical and computer engineering from the Stevens Institute of Technology, Hoboken, NJ, USA, in 2017, and the Ph.D. degree from The State University of New York at Binghamton, Binghamton, New York, USA, in 2020. He is currently pursuing the Ph.D. degree in electrical engineering with the School of Electrical, Computer and Energy Engineering, Arizona State University, Tempe, AZ, USA. His primary research interests include ultra-low-power in-memory computing based on volatile and non-volatile memories, brain-inspired (neuromorphic) computing, and accelerator design for deep neural networks and bioinformatics.

Shaahin Angizi (Senior Member, IEEE) received the Ph.D. degree in electrical engineering from the School of Electrical, Computer and Energy Engineering, Arizona State University (ASU), Tempe, AZ, USA, in 2021. He is currently an Assistant Professor with the Department of Electrical and Computer Engineering, New Jersey Institute of Technology (NJIT), Newark, NJ, USA, and the Director of the Advanced Circuit-to-Architecture Design Laboratory (ACAD-Lab). His primary research interests include ultra-low power in-memory computing based on volatile and non-volatile memories, in-sensor computing, and accelerator design for deep neural network and bioinformatics. He has authored and coauthored more than 80 research papers in top-ranked journals and EDA conferences. He was a recipient of the Best Poster Award of Ph.D. Forum at IEEE/ACM DAC in 2018, two Best Paper Awards of IEEE ISVLSI in 2017 and 2018, and the Best Paper Award of ACM GLSVLSI in 2019.

Jiao Sun received the M.S. degree in electrical engineering from the University of Central Florida in 2018. Her research interests include mRNA isoform quantification with NGS, post-transcriptional regulation, and its application in human disease.



Wei Zhang received the M.S. and Ph.D. degrees in computer science from the University of Minnesota Twin Cities in 2011 and 2015, respectively. He joined the Department of Computer Science, University of Central Florida, Orlando, FL, USA, as an Assistant Professor, in 2017. His primary research interests include the interaction of computational biology and machine learning. He has been focusing on graph-based learning models for biomarker selection and cancer outcome prediction. His other research interests include cancer transcript variants and drug sensitivity prediction. He received NSF CRII Award in 2018.

Deliang Fan (Member, IEEE) received the M.S. and Ph.D. degrees in electrical and computer engineering from Purdue University, West Lafayette, IN, USA, in 2012 and 2015, respectively. He is currently an Associate Professor with the School of Electrical, Computer and Energy Engineering, Arizona State University, Tempe, AZ, USA. His primary research interests include machine learning circuits and algorithm at edge, in-memory computing circuits and architecture, adversarial AI security, and trustworthy. He has authored and coauthored around more than 150 peer-reviewed international journals/conference papers in those areas. He was a receipt of NSF Career Award, Best IP Paper Award of DATE 2022, and Best Paper Award of GLSVLSI 2019, ISVLSI 2018, and ISVLSI 2017. His research works were also nominated as best paper candidate of ASPDAC 2019 and ISQED 2019. He is the Program Chair of ISQED 2023. He is also the Technical Area Chair of DAC 2021, GLSVLSI 2019-2022, and ISQED 2019-2022, and the Financial Chair of ISVLSI 2019.