Exploring DNA Alignment-in-Memory Leveraging Emerging SOT-MRAM

Shaahin Angizi
Arizona State University
Tempe, AZ, USA
sangizi@asu.edu

Wei Zhang
University of Central Florida
Orlando, FL, USA
wzhang.cs@ucf.edu

Deliang Fan
Arizona State University
Tempe, AZ, USA
dfan@asu.edu

ABSTRACT

In this work, we review two alternative Processing-in-Memory (PIM) accelerators based on Spin-Orbit-Torque Magnetic Random Access Memory (SOT-MRAM) to execute DNA short read alignment based on an optimized and hardware-friendly alignment algorithm. We first discuss the reconstruction of the existing sequence alignment algorithm based on BWT and FM-index such that it can be fully implemented leveraging PIM functions. We then transform SOT-MRAM array to a potential computational memory by presenting two different reconfigurable sense amplifiers to accelerate the reconstructed alignment-in-memory algorithm. The cross-layer simulation results show that such PIM platforms are able to achieve a nearly ten-fold and two-fold increases in throughput/power/area measure compared with recent ASIC and processing-in-ReRAM designs, respectively.

CCS CONCEPTS

• **Hardware → Non-volatile memory**; **Analysis and design of emerging devices and systems**;

KEYWORDS

Processing-in-Memory; DNA Alignment; Bioinformatics

ACM Reference Format:
Shaahin Angizi, Wei Zhang, and Deliang Fan. 2020. Exploring DNA Alignment-in-Memory Leveraging Emerging SOT-MRAM. In *Proceedings of the Great Lakes Symposium on VLSI 2020 (GLSVLSI '20), September 7–9, 2020, Virtual Event, China.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3386263.3407590

1 INTRODUCTION

Advances in high-throughput sequencing technologies have enabled accurate and fast generation of large-scale genomic data for each individual, and are capable of measuring molecular activities in cells. Genomic analyses, including mRNA quantification, genetic variants detection, and differential gene expression, promise to help improve phenotype predictions and provide more accurate disease diagnostics [1]. The sequencing data generated from one patient sample consists of tens of millions of short DNA sequences (reads) that range from 50 to thousands *nt* in length. Most genomic pipelines rely on the alignment of sequencing reads with respect to the reference genome [2], which remains to be a time-consuming and technically difficult step. Specifically, the human reference genome is comprised of two twistings, paired strands and each strand carries approximately 3.2 billion nucleotide bases (*A*, *T*, *C*, *G*), and the bases on two strands follow the complementary base pairing rule: *A*-*T* and *C*-*G* [3]. Therefore, the DNA sequence alignment task is becoming to determine the read's likely point of origin on the 3.2 billion base pair (*bp*) reference genome. Although several sequence alignment algorithms have been developed in recent years, the continuously increasing volume of DNA sequencing data still calls for rapid and accurate aligners. Even the most efficient algorithm, such as BWA [2] or Bowtie [4] using Burrows-Wheeler Transformation (BWT), require hours or days to align such large amount of data using powerful CPU/GPU-based systems.

Today's sequencing acceleration platforms, like CPU, GPU [5], ASIC [6–8], and FPGA [9], are mostly based on the Von-Neumann architecture with separate computing and memory components connecting via buses and inevitably consume a large amount of energy in data movement between them. Besides, Processing-in-Memory (PIM) architectures, as a potentially viable way to solve the memory wall challenge [10, 11], have been explored for different applications that lead to remarkable savings in off-chip data communication energy and latency. The PIM platform has become even more intriguing when integrated with emerging Non-Volatile Memory (NVM) technologies, such as Resistive RAM (ReRAM) [3, 12, 13]. The most recent ReRAM-based PIM solutions for short read alignment [12, 14] rely on Ternary Content-Addressable Memory (TCAM) arrays that unavoidably impose significant area and energy overheads to the system [3] due to associative processing dealing with Smith-Waterman (SW)-based algorithms that require many write operations and takes 75% of the ReRAM cells to store the intermediate data [15]. Alternatively, RADAR [12] and AligneR [3] present ReRAM-based PIM architectures that can directly map more efficient algorithms such as BLASTN and FM-index-based searches, respectively. In addition, Magnetic RAM (MRAM) is another promising high performance NVM paradigm, due to its ultra-low switching energy and compatibility with CMOS technology [16–20].

In this work, we review software-hardware alignment-in-memory solutions based on SOT-MRAM platforms presented in our previous works i.e. AlignS [16] and PIM-Aligner [21] to perform DNA sequence alignment efficiently. This effort mainly highlights two

GLSVLSI '20, September 7–9, 2020, Virtual Event, China

ACM ISBN 978-1-4503-7944-1/20/09…$15.00
https://doi.org/10.1145/3386263.3407590

possible circuit/architecture alternatives for DNA alignment-in-memory through reconstruction of the existing sequence alignment algorithm based on BWT and FM-index, with a local data partitioning methodology and mapping technique.

2 BWT-BASED ALIGNMENT-IN-MEMORY READ MAPPING

The BWT of a string is a reversible permutation of the characters in the string. Short read alignment algorithms (e.g., BWA [2] and Bowtie [4]) take all the advantages of BWT and index the large reference genome-S to do the read alignment efficiently. Exact alignment finds all occurrences of the m-bp short-read R in the n-bp reference genome-S. Fig. 1 gives an intuitive example of such alignment of a sample read-$R = CTA$ to a sample reference $S = TGCTA\$$ extracted from a gene, where \$ denotes the end of a sequence. BW matrix is constructed by circulating string S and then lexicographically sorting them. Thus, the Suffix Array (S_A) of a reference genome-S is a lexicographically-sorted array of the suffixes of S, where each suffix is represented by its position in S. In this way, BWT of the reference-S is given by the last column in the BW matrix, $BWT(S) = ATGTC\$$.

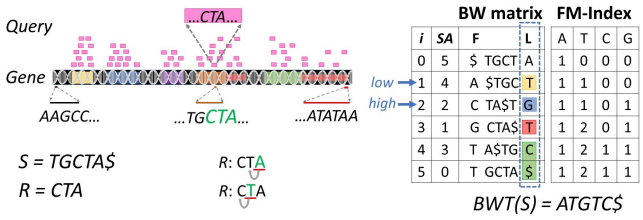

Figure 1: Short read alignment concept.

The FM-Index is then built on top of BWT providing the occurrence information of each symbol in BWT. The S_A interval (*low*, *high*) covers a range of indices where the suffixes have the same prefix. Then a backward search of the matched positions in the reference genome-S is executed for each short read-R starting from the rightmost nucleotide (A in Fig. 1). During the backward search, 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 [2]. Thus, the result of read searching is represented as an S_A interval. At the end of search, if $low < high$, R has found a match in S. Conversely, if $low \geq high$, it has failed to find a match. Such alignment algorithm complexity is linearly proportional to the number of nucleotides in a read ($O(m)$) in contrast to dynamic programming algorithms such as Smith-Waterman (SW) with $O(nm)$ complexity [22]. Backtracking can simply extend the BWT technique to allow mismatches to support approximate alignment. In this approach, the DNA short read is permuted using edit operations (substitutions, insertions or deletions).

The presented DNA exact alignment-in-memory algorithm is based on BWT and FM-Index sequencing algorithm [2], but optimized using three different PIM functions, i.e. MEM, XNOR_Match, and IM_ADD, which will be detailed later. As the first step of such process, shown in Fig. 2a, some important tables are needed to be pre-computed based on reference genome-S. However, it is just a one-step computation and only BWT, Marker Table (M_T), and

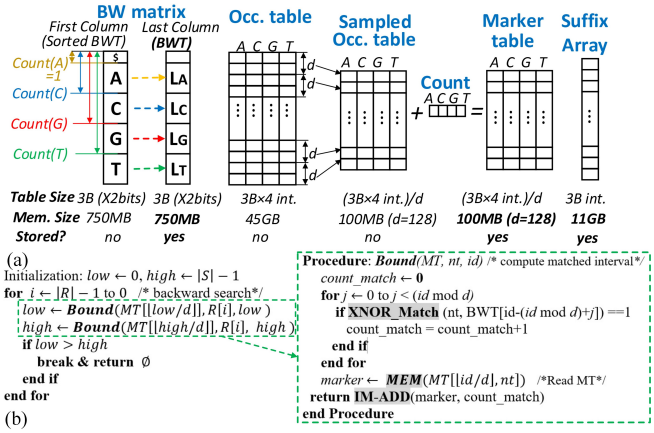

Figure 2: (a) Pre-computation needed in PIM's alignment algorithm, (b) The *Bound* procedure implementation.

S_A will be stored in the PIM platform, which will consume ~12GB of memory space. To enable fast memory access and parallel in-memory computing, these data has to be reconstructed and saved into different memory arrays, banks and chips. Such data reconstruction and mapping methodology will be discussed in the next section. In Fig. 2a, $Count(nt)$ represents the number of nucleotides in the first column of BW matrix that are lexicographically smaller than the nucleotide-nt. It only contains 4 elements for DNA sequence computation. The Occurrence (Occ.) table, also called FM-index, is built upon the BWT, where each element-$Occ[i, nt]$ indicates the number of occurrences of nucleotide-nt in the BWT from position 0 to $i-1$. Due to its large size, it is sampled every d positions (bucket width) to construct another Sampled Occ-table. Thus, the table size is reduced by a factor of d. Then M_T is constructed by element-wise addition of Sampled Occ-table with $Count(nt)$, which leads to the same size as Sampled Occ-Table. M_T contains the matched position of the nucleotides in BWT in the First Column and helps the PIM platform to efficiently retrieve the values of *low* and *high* in each iteration. As shown in Fig. 2b, the read searching operation is mainly implemented through the presented $Bound(M_T, nt, id)$ procedure performed on BWT, which computes the updated interval bound (either *low* or *high*) value from M_T with bucket width d and input index-id. Such procedure is iteratively used in every step of 'for' loop and our PIM platforms must be designed in a way to handle such computation-intensive load through comparison and summing the current 'marker' value with the occurrence counting result of the needed nucleotides between checkpoint position and remaining positions in BWT.

3 PIM ARCHITECTURES

The PIM accelerators are mainly developed on top of main memory architecture. To run the above discussed DNA exact algorithm supporting backtracking with a hardware PIM platform, we use the block level accelerator architecture shown in Fig. 3. It includes BWT-based mapping based on iteratively-used *Bound* procedure and S_A and M_T query as the crucial operations. A Digital Processing Unit (DPU) is associated with the PIM platforms to control the entire process through different steps. The DPU takes the reference genome-S and number of mismatches-z as the inputs and accordingly adjusts the controller unit to govern timing and data

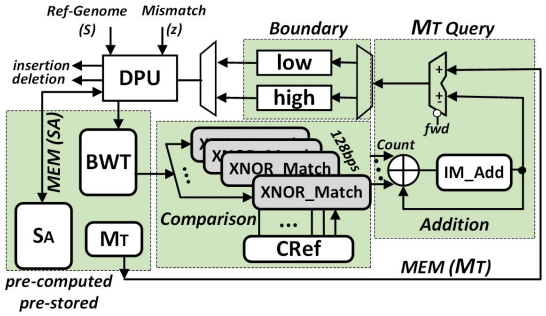

Figure 3: The block level architecture of the presented PIM platforms to support the alignment algorithm with backtracking.

flow of the alignment task. The index-*low* and index-*high* boundaries are initialized to the length of the reference DNA, 0 and N, respectively. As mentioned earlier, the possible locations of the suffixes candidate are indicated with the range of index-*low* and index-*high*. The backtracking is then performed in each iteration based on the discussed alignment algorithm. To implement the *Bound* procedure totally within memory, we need three functions, i.e. *MEM* (memory read), *XNOR_Match* (Comparison-XNOR2), and *IM_ADD* (Addition-add), as highlighted in the algorithm and Fig. 3. *MEM* function is to access data in the saved M_T or S_A based on the provided index. *XNOR_Match* is to conduct parallel in-memory XNOR2 logic to determine if current input-*nt* matches with BWT elements stored in the whole word-line. *IM_ADD* is to conduct 32-bit integer (index range) addition operation within memory to enable fast 'marker+count_match' computation without need to send to other computing units. To reduce the computation load by the down sampling scheme for reconstructing Occ. table, PIM platforms exploit multiple *XNOR_Match* modules in parallel. Given a 512×256 memory sub-array, 128-bps could be compared to maximize the query speed for 2-bit DNA bases. This will be elaborated in Section 4.

Each MRAM chip is divided into multiple Banks. Banks within the same chip typically share I/O, buffer and banks in different chips working in a lock-step manner. Each bank consists of multiple memory matrices (mats) [23]. The general mat organization of PIM accelerators is shown in Fig. 4a. Each mat consists of multiple computational memory sub-arrays connected to a Global Row Decoder (GRD) and a shared Global Row Buffer (GRB). According to the physical address of operands within memory, the PIM's Controller (Ctrl) is able to configure the sub-arrays to perform data-parallel intra-sub-array computations. Moreover, every two sub-arrays share a Local Row Buffer (LRB).

Fig. 4b depicts the PIM sub-array architecture based on SOT-MRAM. This architecture mainly consists of Write Driver (WD), Memory Row Decoder (MRD), Memory Column Decoder (MCD), reconfigurable Sense Amplifier (SA), and can be adjusted by Ctrl unit to work in dual-mode that perform both memory write/read and bit-line computing. SOT-MRAM device is a composite structure of Spin Hall Metal (SHM) and Magnetic Tunnel Junction (MTJ) [24, 25]. The resistance of MTJ with parallel magnetization in both magnetic layers (data-'0') is lower than that of MTJ with anti-parallel magnetization (data-'1'). Each SOT-MRAM cell located in computational sub-arrays 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 operations based on reconfigurability of memory SAs. Here, we mainly discuss two designs based on block level architecture called *AlignS* [16] and *PIM-Aligner* [21].

3.1 Design I: AlignS

The key idea to perform memory read and bit-line computing in our PIM platforms (here, *AlignS* [16]) is to choose different thresholds (references) when sensing the selected memory cell(s). The presented *AlignS*'s reconfigurable SA, as depicted in Fig. 4c, consists of two sub-SAs and totally four reference-resistance branches that can be selected by control bits (C_{MAJ}, C_{AND2}, C_{OR2}, C_{Mem}) by the sub-array's Ctrl to realize the memory and computation schemes. Later, the output is routed through two mux controllers (C_{muxI} and C_{muxII}) to either LRBout1 and LRBout2. Such reconfigurable SA is especially optimized to accelerate two read alignment's intensive operations, i.e. 2-input XNOR and addition as well as typical memory read operation. Therefore, there are only four available control-bit sequences (shown in Fig. 4c) that provide an efficient control circuitry for *AlignS*.

MEM: For memory *read*, a read current flows from the selected cell to ground, generating a sense voltage (V_{sense}) at the input of the second SA in Fig. 4c, which is compared with memory mode reference voltage activated by C_M ($V_{sense,P}$<$V_{ref,M}$<$V_{sense,AP}$). If the path resistance is higher (/lower) than R_M (memory reference resistance), i.e. R_{AP} (/R_P), then the SA produces High (/Low) voltage indicating logic '1' (/'0'). The idea of voltage comparison for memory read is shown in Fig. 5a.

XNOR_Match: To realize XNOR2 in-memory logic, every two bits stored in the identical column can be selected employing the MRD [26] and sensed simultaneously, as depicted in Fig. 4b-top. Then, the equivalent resistance of such parallel connected cells and their cascaded access transistors are compared with two programmable references by SAs (R_{AND}, R_{OR}). Through selecting different reference resistances, the sub-SAs can perform basic 2-input in-memory Boolean functions (i.e. AND2/NAND2 and OR2/NOR2), e.g. to realize AND2 operation, R_{AND} is set at the midpoint of $R_{AP}//R_P$ ('1','0') and $R_{AP}//R_{AP}$ ('1','1'). Accordingly, as shown in 4c, we formed a capacitive voltage divider after OR2 and NAND2 outputs driving a CMOS inverter (with low-Vth PMOS and high-Vth NMOS) to realize NAND2 function, thereby enabling a multi-kilobyte-wide bitwise XNOR2 of two rows in *AlignS*'s sub-arrays. Note that, dual-threshold technique can eliminate the leakage current through a transistor, thereby decreasing leakage power consumption while maintaining performance [27, 28]. The idea of voltage comparison between V_{sense} and V_{ref} to realize AND2/NAND2 and OR2/NOR2 is shown on Fig. 5b.

IM_ADD: *AlignS*'s sub-array can perform addition/subtraction (add/sub) operation quite efficiently. The carry-out of the full-adder can be directly produced by MAJ3 function (Carry in Fig. 4c) just by setting C_{MAJ} to '1' in a single memory cycle. As shown in Fig. 5c, to perform MAJ3 operation, R_{MAJ} is set at the midpoint of $R_P//R_P//R_{AP}$ ('0','0','1') and $R_P//R_{AP}//R_{AP}$ ('0','1','1'). Meanwhile, the existing latch in LRB (Fig. 4c) is equipped with additional NOT and XOR2 gates to first store intermediate carry outputs and then perform the summation of next bits using two XOR2 gates (implementing XOR3). Now, assume A, B, and C operands (Fig. 4b), the 3-

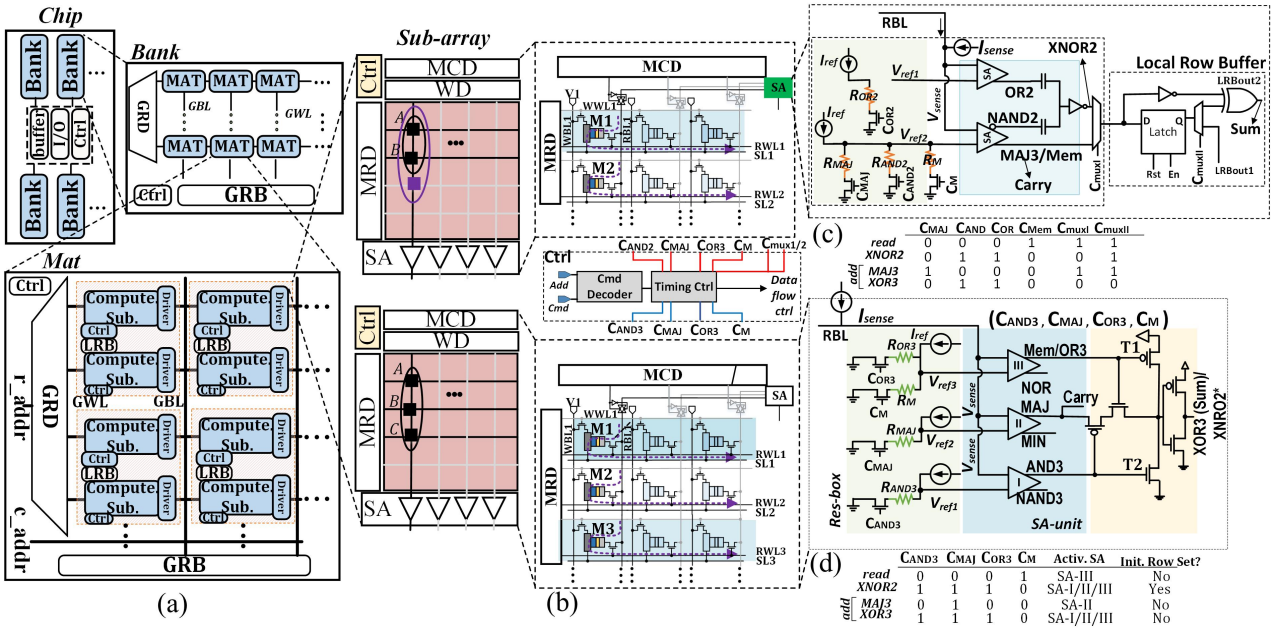

Figure 4: (a) The PIM memory organization, (b) Block level scheme of computational sub-array and SOT-MRAM realization, (c) *AlignS*'s reconfigurable SA [16], (d) *PIM-Aligner*'s reconfigurable SA [21].

and 2-input in-memory logic schemes can generate Carry(/Borrow) and Sum (/Difference), respectively, in two consecutive cycles. The Ctrl's configuration for such add operation is shown in Fig. 4c.

3.2 Design II: PIM-Aligner

The *PIM-Aligner*'s reconfigurable SA [21], as depicted in Fig. 4d, consists of three sub-SAs and totally four reference-resistance branches that can be selected by enable control bits (C_{AND3}, C_{MAJ}, C_{OR3}, C_M) by the sub-array's Ctrl to realize the memory read and computation schemes, as tabulated in the table in Fig. 4d. Such reconfigurable SA could implement memory read and one-threshold based logic functions only by activating one enable at a time e.g. by setting C_{AND3} to '1', 3-input AND/NAND logic can be readily implemented between operands located in the same bit-line.

MEM: The memory *read* operation is similar to that of Design I in Fig. 5a.

XNOR_Match: *PIM-Aligner*'s SA exploits a unique circuit design that allows single-cycle implementation of XOR3 in-memory logic. To realize XOR3 in-memory logic, every three bits stored in the identical column can be selected employing the MRD [26] and sensed simultaneously, as depicted in Fig. 4b-below. Then, the equivalent resistance of such parallel connected cells and their cascaded access transistors are compared with three programmable references by SAs (R_{AND3}, R_{MAJ}, R_{OR3}). Through selecting these reference resistances simultaneously, the sub-SAs can perform basic 3-input in-memory Boolean functions (i.e. AND3,MAJ, OR3). The idea of voltage comparison between V_{sense} and V_{ref} to realize these functions is shown on Fig. 5c. After *SA-unit*, we used six control transistors to realize XOR3 function. Assuming one row in memory sub-array initialized to one, XNOR2 can be readily implemented in a single memory cycle out of XOR3 function. Therefore, every memory sub-array can potentially perform *XNOR_Match* function in DNA algorithm.

IM_ADD: The carry-out of the full-adder can be directly produced by MAJ function (Carry in Fig. 4d) just by setting C_{MAJ} to '1' in a single memory cycle. Now, assume M1, M2, and M3 operands (Fig. 4b-below), the *PIM-Aligner* can generate Carry-MAJ and Sum-XNOR3 in-memory logics in a single memory cycle. The Ctrl's configuration for such add operation is tabulated in Fig. 4d.

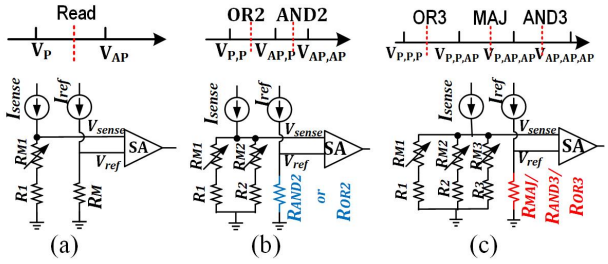

Figure 5: Reference comparison to realize in-memory operations: (a) Memory read, (b) 2-input operations in AlignS [16], (c) 3-input operations in PIM-Aligner [21]. Note: MAJ function is applicable to both PIM designs.

4 CORRELATED AND LOCALIZED COMPUTATION

4.1 Partitioning

The pre-computed tables (BWT, M_T, and S_A) require a large memory space, therefore, to fully leverage *AlignS* and *PIM-Aligner*'s parallelism, and maximize alignment throughput, we come up with a partitioning, mapping and pipeline design. Given a BWT index range, the accessed memory region of M_T and BWT could be readily

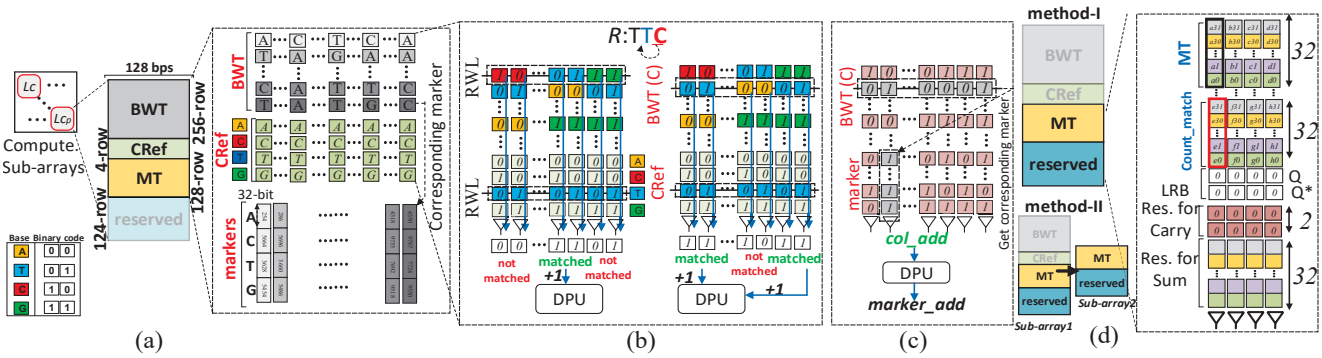

Figure 6: (a) The PIM sub-array partitioning technique for comparison and addition operations, (b) Parallel comparison operation (*XNOR_Match*), (c) *MEM* function to retrieve marker_add, (d) *IM_ADD* function with two methods.

predicted and computation could be localized if we store such correlated region into the same memory sub-array. The correlated data partitioning and mapping methodology, as shown in Fig. 6, locally stores correlated regions of BWT and M_T vectors in the same memory sub-array to enable fully local computation (i.e. *XNOR_Match* and *IM_ADD* completely within the same sub-array without inter-bank/chip communication). To do it, each PIM sub-array (512×256) is spilt into four zones to save four different data types, i.e. BWT, CRef, M_T, and reserved space for *IM_ADD* (Fig. 6a). First, 256 rows are filled with the corresponding BWT, where each row stores up to 128 bps (encoded by 2 bits). Besides, 4 nucleotide computational reference vectors (*CRef*) are initialized, in which each vector gives one type of nucleotide with vector size of number of bits in one word-line. CRef is designed to enable fully parallel match operation-*XNOR_Match*. Next to it, the value of markers (M_T) is check-pointed every d (=128) positions (one row), and vertically stored to keep the size in check within the platform. Hence, 256 columns are allocated for storing M_T, each storing 4-byte value for bps (128-bit). After partitioning, starting from the rightmost symbol in R, the *Bound* procedure runs and returns *low* and *high* for next symbol.

4.2 Mapping and Computation

Considering current input nucleotide is T and input index as id (in Fig. 6b), *AlignS* and *PIM-Aligner* convert the BWT index into the corresponding memory WL and BL addresses storing data $BWT[id - (id \bmod d)]$ to $BWT[id]$. Then, such bits and corresponding CRef-T can implement the parallel comparison operations (*XNOR_Match*). If the XNOR output is '1' (a match is found), DPU's embedded counter counts up to eventually compute *count_match* for next operation. Fig. 6b intuitively shows the *XNOR_Match* procedure to locate Ts in a sub-array. When counting is done, the sub-array returns the *count_match* and marker address (*marker_add*), shown in Fig. 6c. The correlated data partitioning methodology guarantees the read of *marker* value (*MEM*) is always a local memory access within the same memory array (Fig. 6c). Now, the *marker* and just computed and transposed *count_match* are buffered in M_T and reserved memory spaces, respectively, as shown in Fig. 6d. To further implement *IM_ADD* function, we present two distinct hardware-friendly methods; method-I performs the bit-line addition within the same computational sub-array based on in-memory addition operation though it degrades the system performance as other sub-array resources (*MEM* and *XNOR_Match*) are not used. To alleviate this issue, method-II essentially duplicates the number of sub-arrays, where only in-memory addition computation is transferred to a second sub-array.

5 PERFORMANCE EVALUATION

5.1 Evaluation Framework

To evaluate the performance of *AlignS* and *PIM-Aligner*, a comprehensive device-to-architecture evaluation framework with two in-house simulators were developed. At the device level, we jointly used the Non-Equilibrium Green's Function (NEGF) and Landau-Lifshitz-Gilbert (LLG) with spin Hall effect equations to model SOT-MRAM bit-cell [24]. For the circuit level, a Verilog-A model of 2T1R SOT-MRAM device was developed to co-simulate with the interface CMOS circuits in Cadence Spectre and SPICE. We used 45nm North Carolina State University (NCSU) Product Development Kit (PDK) library in SPICE to analyze the presented design and achieve the performance. Besides, we built an architectural-level simulator on NVSim [29]. It can change the configuration files (.cfg) corresponding to different array organization and report performance for PIM operations based device/circuit level data. Then we fed the performance data to a behavioral simulator based on Matlab to calculate the latency and energy that each platform spends on alignment task based on the algorithms. We perform an extensive comparison with the counterpart computing platforms including SW-based Darwin [7], ReCAM [30] and RaceLogic [6], as well as FM-Index-based platforms including Soap3-dp [5] on GPU, FPGA [9], ASIC [8], AligneR [3]. In the interest of limited space, we refer the readership to the papers for the detailed configuration of each accelerator. Note that, to perform short read alignment on GPU, we used Soap3 [5] considering only reads with ≤2 mismatches. We re-implemented, ReRAM-, SOT-MRAM, and CAMs with NVSim [29]. For evaluation, we generated 10 million 100-bps short read queries via ART simulator [31] and aligned them to the human genome Hg19 with different computing platforms. Note that the population variation and genome error rate were set to 0.1% and 0.2%.

5.2 Results

We report the estimated performance of short read alignment task on our presented platforms compared with various acceleration solutions in Table 1. From the throughput point of view, Race Logic [6] and Darwin [7] platforms show the best performance compared to

Table 1: Performance of short read alignment accelerators.

	GPU	ASIC	FPGA	AligneR	Darwin	Race	ReCAM	AlignS	PIM-Aligner
Throughput (query/sec)	150K	379K	1.5M	483K	5.7M	8.3M	177K	4.7M	4.9M
Power (W)	393	3.1	247	1.86	442	97	6.6K	11.4	18.4
Throughput/Power (query/sec/W)	581.3	122K	6.1K	259.6K	12.8K	85K	26.81	412.28K	266K
Throughput/Power/Area (query/sec/W/mm^2)	0.39	547	0.42	7.2K	0.47	47	0.24	6.6K	4.83K

others. However, *PIM-Aligner* and then *AlignS* achieve the highest throughput compared to others such as GPU, ASIC, FPGA, ReCAM, and AligneR due to their massively-parallel and local computational scheme. In terms of power consumption, we observe that ASIC design [8] and ReRAM-based AligneR [3] consume the least power compared to other designs, where *AlignS* and *PIM-Aligner* stand as the third- and fourth-best power-efficient design.

We further investigate the existing trade-offs between power, throughput, and area Table 1. Such trade-off can be better understood by correlated parameters as tabulated in Table 1. Based on this table, we observe that *AlignS* outperforms different accelerators w.r.t. throughput/power. *AlignS* improves short read alignment's throughput per Watt by 4.8× over the best SW-based accelerator, Race Logic [6], and ∼1.6×, 3.4×, 67.5× over AligneR [3], ASIC [8], and FPGA [9] acceleration solutions, respectively. *PIM-Aligner* achieves the second highest throughput/power ratio due to the its intrinsic three SAs scheme compared with *AlignS*. Table 1 also estimates throughput per power per area for various accelerators. Considering the area factor, we observe that *AlignS* and *PIM-Aligner* can improve read alignment performance significantly over all the other solutions except AligneR. *AlignS* improves the throughput per Watt per mm^2 by ∼12× compared to the ASIC accelerator. Therefore, *AlignS* and *PIM-Aligner*'s parallel computing schemes can be leveraged to accelerate short read alignment and provide ultra-high internal bandwidth.

6 CONCLUSION

In this paper, we studied two promising Processing-in-Memory (PIM) accelerators based on SOT-MRAM to run DNA short read alignment task leveraging on an optimized and PIM-friendly alignment algorithm. Our cross-layer simulation results and comparison show that such PIM platforms improve the alignment task throughput per power per area compared to promising GPU, ASIC, FPGA, and recent processing-in-ReRAM designs.

ACKNOWLEDGEMENTS

This work is supported in part by the National Science Foundation under Grant No.2005209, No.2003749, No. 1755761 and Semiconductor Research Corporation nCORE.

REFERENCES

[1] H. Li and N. Homer, "A survey of sequence alignment algorithms for next-generation sequencing," *Briefings in bioinformatics*, vol. 11, no. 5, pp. 473–483, 2010.
[2] H. Li and R. Durbin, "Fast and accurate short read alignment with burrows–wheeler transform," *bioinformatics*, vol. 25, pp. 1754–1760, 2009.
[3] F. Zokaee *et al.*, "Aligner: A process-in-memory architecture for short read alignment in rerams," *IEEE Computer Architecture Letters*, 2018.
[4] B. Langmead *et al.*, "Ultrafast and memory-efficient alignment of short dna sequences to the human genome," *Genome biology*, vol. 10, p. R25, 2009.
[5] R. Luo *et al.*, "Soap3-dp: fast, accurate and sensitive gpu-based short read aligner," *PloS one*, vol. 8, p. e65632, 2013.
[6] A. Madhavan *et al.*, "Race logic: A hardware acceleration for dynamic programming algorithms," in *ACM SIGARCH Computer Architecture News*, vol. 42, 2014.
[7] Y. Turakhia *et al.*, "Darwin: A genomics co-processor provides up to 15,000 x acceleration on long read assembly," in *23rd ASPLOS*. ACM, 2018, pp. 199–213.
[8] Y.-C. Wu *et al.*, "A 135-mw fully integrated data processor for next-generation sequencing," *IEEE TBioCAS*, vol. 11, pp. 1216–1225, 2017.
[9] J. Arram *et al.*, "Leveraging fpgas for accelerating short read alignment," *IEEE/ACM TCBB*, vol. 14, pp. 668–677, 2017.
[10] S. Angizi and D. Fan, "Redram: A reconfigurable processing-in-dram platform for accelerating bulk bit-wise operations," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2019, pp. 1–8.
[11] S. Angizi, Z. He, F. Parveen, and D. Fan, "Rimpa: A new reconfigurable dual-mode in-memory processing architecture with spin hall effect-driven domain wall motion device," in *2017 IEEE Computer Society annual symposium on VLSI (ISVLSI)*. IEEE, 2017, pp. 45–50.
[12] W. Huangfu *et al.*, "Radar: a 3d-reram based dna alignment accelerator architecture," in *55th DAC*. ACM, 2018, p. 59.
[13] S. Angizi and D. Fan, "Accelerating bulk bit-wise x (n) or operation in processing-in-dram platform," *arXiv preprint arXiv:1904.05782*, 2019.
[14] S. K. Khatamifard *et al.*, "A non-volatile near-memory read mapping accelerator," *arXiv preprint arXiv:1709.02381*, 2017.
[15] L. Yavits *et al.*, "Resistive associative processor," *IEEE Computer Architecture Letters*, vol. 14, pp. 148–151, 2015.
[16] S. Angizi, J. Sun, W. Zhang, and D. Fan, "Aligns: A processing-in-memory accelerator for dna short read alignment leveraging sot-mram," in *2019 56th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2019, pp. 1–6.
[17] Z. I. Chowdhury, M. Zabihi, S. K. Khatamifard, Z. Zhao, S. Resch, M. Razaviyayn, J.-P. Wang, S. S. Sapatnekar, and U. R. Karpuzcu, "A dna read alignment accelerator based on computational ram," *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits*, 2020.
[18] S. Angizi, Z. He, and D. Fan, "Dima: a depthwise cnn in-memory accelerator," in *2018 ICCAD*. IEEE, 2018, pp. 1–8.
[19] S. Angizi, Z. He, A. S. Rakin, and D. Fan, "Cmp-pim: an energy-efficient comparator-based processing-in-memory neural network accelerator," in *Proceedings of the 55th Annual Design Automation Conference*. ACM, 2018, p. 105.
[20] S. Angizi, Z. He, F. Parveen, and D. Fan, "Imce: Energy-efficient bit-wise in-memory convolution engine for deep neural network," in *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2018, pp. 111–116.
[21] S. Angizi, J. Sun, W. Zhang, and D. Fan, "Pim-aligner: A processing-in-mram platform for biological sequence alignment," in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2020, pp. 1265–1270.
[22] S. Canzar *et al.*, "Short read mapping: An algorithmic tour," *Proceedings of the IEEE*, pp. 436–458, 2017.
[23] S. Angizi and D. Fan, "Graphide: A graph processing accelerator leveraging in-dram-computing," in *Proceedings of the 2019 on Great Lakes Symposium on VLSI*, 2019, pp. 45–50.
[24] X. Fong, Y. Kim, K. Yogendra, D. Fan, A. Sengupta, A. Raghunathan, and K. Roy, "Spin-transfer torque devices for logic and memory: Prospects and perspectives," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 1, pp. 1–22, 2015.
[25] S. Angizi, Z. He, A. Awad, and D. Fan, "Mrima: An mram-based in-memory accelerator," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 5, pp. 1123–1136, 2019.
[26] S. Li, C. Xu, Q. Zou, J. Zhao, Y. Lu, and Y. Xie, "Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories," in *Proceedings of the 53rd Annual Design Automation Conference*, 2016, pp. 1–6.
[27] S. V. Kosonocky *et al.*, "Enchanced multi-threshold (mtcmos) circuits using variable well bias," in *ISLPED*, 2001, pp. 165–169.
[28] H. Ozdemir, A. Kepkep, B. Pamir, Y. Leblebici, and U. Cilingiroglu, "A capacitive threshold-logic gate," *IEEE Journal of Solid-State Circuits*, vol. 31, no. 8, pp. 1141–1150, 1996.
[29] X. Dong *et al.*, "Nvsim: A circuit-level performance, energy, and area model for emerging non-volatile memory," in *Emerging Memory Technologies*. Springer, 2014, pp. 15–50.
[30] R. Kaplan *et al.*, "A resistive cam processing-in-storage architecture for dna sequence alignment," *IEEE Micro*, vol. 37, pp. 20–28, 2017.
[31] W. Huang *et al.*, "Art: a next-generation sequencing read simulator," *Bioinformatics*, vol. 28, pp. 593–594, 2011.