AlignS: A Processing-In-Memory Accelerator for DNA Short Read Alignment Leveraging SOT-MRAM

Shaahin Angizi[†], Jiao Sun[‡], Wei Zhang[‡] and Deliang Fan[†]
[†] Department of Electrical and Computer Engineering, University of Central Florida, Orlando, FL 32816
[‡] Department of Computer Science, University of Central Florida, Orlando, FL 32816
{angizi,jiao.sun}@knights.ucf.edu,{wzhang.cs,dfan}@ucf.edu

ABSTRACT

Classified as a complex big data analytics problem, DNA short read alignment serves as a major sequential bottleneck to massive amounts of data generated by next-generation sequencing platforms. With Von-Neumann computing architectures struggling to address such computationally-expensive and memory-intensive task today, Processing-in-Memory (PIM) platforms are gaining growing interests. In this paper, an energy-efficient and parallel PIM accelerator (*AlignS*) is proposed to execute DNA short read alignment based on an optimized and hardware-friendly alignment algorithm. We first develop *AlignS* platform that harnesses SOT-MRAM as computational memory and transforms it to a fundamental processing unit for short read alignment. Accordingly, we present a novel, customized, highly parallel read alignment algorithm that only seeks the proposed simple and parallel in-memory operations (i.e. comparisons and additions). *AlignS* is then optimized through a new correlated data partitioning and mapping methodology that allows local storage and processing of DNA sequence to fully exploit the algorithm-level's parallelism, and to accelerate both exact and inexact matches. The device-to-architecture co-simulation results show that *AlignS* improves the short read alignment throughput per Watt per mm^2 by ~12× compared to the ASIC accelerator. Compared to recent FM-index-based ReRAM platform, *AlignS* achieves 1.6× higher throughput per Watt.

1 INTRODUCTION

Powered by the high-throughput genomic technologies, the new DNA sequencing method is able to determine the accurate order of nucleotides (*nt*) along genomes, and capable of measuring molecular activities in cells. It empowers disease diagnostics and other aspects of medical care, including tailor patient treatment and prenatal testing [1]. The sequence data generated from one patient sample consists of tens of millions short DNA sequences (reads) that range from 50 to 500 *nt* in length. These short reads do not come with position information, and we do not know what part of the chromosome/genome they came from. Thus, the short reads must be aligned to the reference genome before most Genomic analyses can begin. However, the reference genome is really big. It contains two twisting, paired strands and each strand carries approximately 3 billion nucleotide bases (A, T, C, G) in human, and the bases on two strands are paired specifically: A-T and C-G [2]. Therefore, the DNA sequence alignment task for one sample is becoming to map the tens of millions of short reads to a 3 billion base pair (*bp*) reference genome with 1-2 mismatches allowed on each short read. Several sequence alignment algorithms have been developed during the last decade. However, even the most efficient algorithm such as BWA [3] or Bowtie [4] using Burrows-Wheeler Transformation (BWT) requires hours or days to align such large amount of data using DNA sequencing machine and even very powerful CPU/GPU-based computing systems. Therefore, the genomic information from DNA sequencing data cannot be widely applied for disease diagnosis and prognosis as the other physiological data in clinics and hospitals.

From computing hardware perspective, today's sequencing acceleration solutions including CPU, GPU [5], ASIC [1, 6, 7], and FPGA [8] 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. In the last two decades, Processing-in-Memory (PIM) architectures, as a potentially viable way to solve the memory wall challenge, have been well explored for different applications [9]. The key concept behind PIM is to realize logic computation within memory to process data by leveraging the inherent parallel computing mechanism and exploiting large internal memory bandwidth. It could lead to remarkable savings in off-chip data communication energy and latency. The PIM architecture has become even more intriguing when integrated with emerging Non-Volatile Memory (NVM) technologies, such as Resistive RAM (ReRAM) [2, 10]. ReRAM offers more packing density (~ 2 – 4×) than DRAM, but they suffer slower and more power hungry writing operation. The most recent ReRAM-based PIM solutions for short read alignment [10, 11] rely on Ternary Content-Addressable Memory (TCAM) arrays that unavoidably impose significant area and energy overheads to the system [2] 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 [12]. Alternatively, RADAR [10] and AligneR [2] present ReRAM-based PIM architectures that can directly map more efficient algorithms such as BLASTN and FM-index-based searches, respectively. In emerging NVM technologies, Magnetic RAM (MRAM) is another promising high performance paradigm, due to its ultra-low switching energy, non-volatility, superior endurance, and compatibility with CMOS technology [13, 14].

In this work, we intertwine the innovation from both architecture and algorithm perspectives: (1) We first design a reconfigurable PIM architecture based on Spin Orbit Torque MRAM (SOT-MRAM), *AlignS*, based on a set of novel microarchitectural and circuit-level

DAC '19, June 2–6, 2019, Las Vegas, NV, USA
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6725-7/19/06…$15.00
https://doi.org/10.1145/3316781.3317764

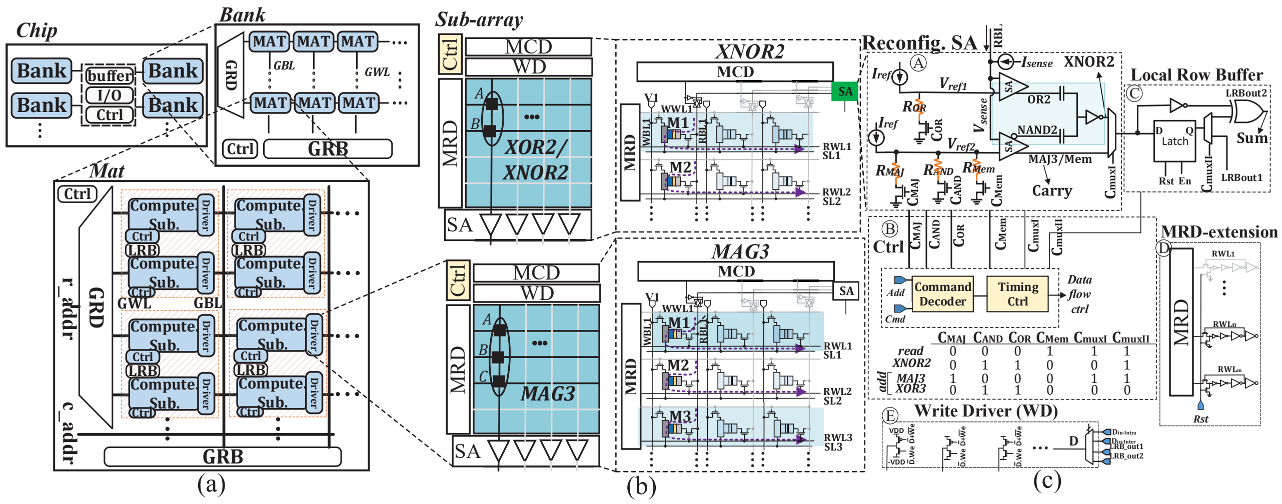


Figure 1: (a) The *AlignS* memory organization, (b) Block level scheme of computational sub-array and SOT-MRAM realization of `XNOR2` **and** `MAJ3` **in-memory logic functions, (c) *AlignS*'s peripheral circuitry.**

schemes that position *AlignS* as a massive data-parallel computational unit for short read alignment; (2) We investigate optimization of fast BWT and FM-index based DNA sequence alignment algorithm from hardware perspective to fully exploit *AlignS*'s parallelism to accelerate DNA alignment; (3) We design a dense data mapping and partitioning scheme to process the indices locally and handle various length DNA sequences; (4) We extensively evaluate and compare *AlignS*'s efficiency compared with state-of-the-art short read alignment accelerators i.e. GPU, ASIC, TCAM, etc.

2 ALIGNS ARCHITECTURE

AlignS is designed to be an independent, high-performance, energy-efficient accelerator based on main memory architecture. The main memory rank is a set of MRAM chips. Each 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). The general mat organization of *AlignS* is shown in Fig. 1a. 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, *AlignS*'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. 1b depicts the presented PIM sub-array architecture based on SOT-MRAM. This architecture mainly consists of Write Driver (WD) (Fig. 1c Ⓔ), 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) [15]. 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.

The key idea to perform memory read and bit-line computing in *AlignS* is to choose different thresholds (references) when sensing the selected memory cell(s). The proposed reconfigurable SA, as depicted in Fig. 1c Ⓐ, consists of two sub-SAs and totally four reference-resistance branches that can be selected by control bits (C_{MAJ}, C_{AND}, C_{OR}, 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. 1c Ⓑ) that provide an efficient control circuitry for *AlignS*.

Memory Mode: To *write* '1' (/'0') in any of the SOT-MRAM cells, e.g. in the cell of 1st row and 1st column (M1), the WD (V1) connected to WBL1 is set to positive (/negative) write voltage. This allows sufficient charge current flows from V1 to ground (/ground to V1), leading to MTJ resistance in High-R_{AP} (/Low-R_P). For memory *read*, a read current flows from the selected cell to ground, generating a sense voltage (V_{sense}) at the input of SA, which is compared with memory mode reference voltage activated by C_{Mem} ($V_{sense,P} < V_{ref,M} < V_{sense,AP}$). If the path resistance is higher (/lower) than R_{Mem} (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. 2a.

Bit-line Computing Mode: The computational sub-array of *AlignS* is optimized to perform two bulk bit-wise in-memory logic operations between the operands located in the same bit-line. To realize `XNOR2` in-memory logic, every two bits stored in the identical column can be selected employing the MRD [16] and sensed simultaneously, as depicted in Fig. 1b. Then, the equivalent resistance of such parallel connected cells and their cascaded access transistors are compared with two programmable references by SA (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 1c Ⓐ, 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 [17, 18]. The idea of voltage comparison between V_{sense} and V_{ref} to realize AND2/NAND2 and OR2/NOR2 is shown on Fig. 2b.

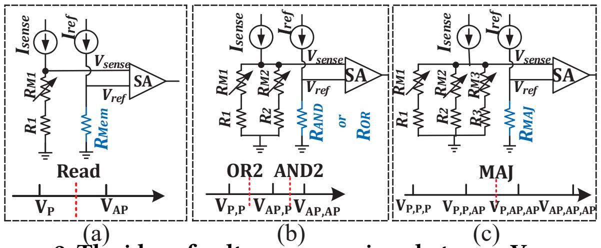


Figure 2: The idea of voltage comparison between V_{sense} and V_{ref} for (a) memory read, (b) (N)AND2, (N)OR2, (c) MAJ3.

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. 1c Ⓐ) just by setting C_{MAJ} to '1' in a single memory cycle. As shown in Fig. 2c, 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. 1c Ⓒ) 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. 1b), the 3- 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. 1c Ⓑ. To validate the variation tolerance of the sensing circuit, we have performed Monte-Carlo simulation with 10000 trials. A σ = 2% variation is added to the Resistance-Area product (RA_P), and a σ = 5% process variation is added on the Tunneling MagnetoResistive (TMR). The simulation result of V_{sense} distributions in Fig. 3 shows the sense margin for the memory read, two and three fan-in sense-based operations. It can be seen that sense margin gradually reduces when increasing the number of fan-ins (selected cells for computation). To avoid logic failure and guarantee the SA output's reliability, we have limited the number of sensed cells to three. In order to provide a larger sense margin for MAJ3 operation, we increased SOT-MRAM cell's t_{ox} from 1.5nm to 2nm leading to ~45mV increase in the sense margin which considerably enhances the reliability.

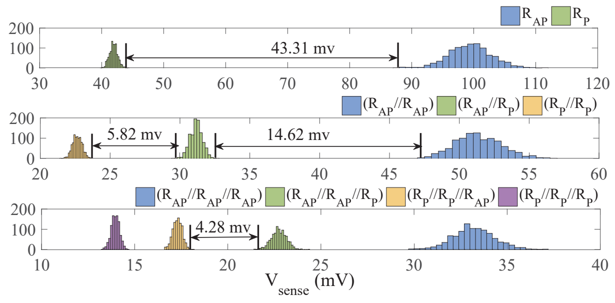


Figure 3: Monte-Carlo simulation of V_{sense} for (top) memory read, and bit-line computing (middle) 2-row (down) 3-row.

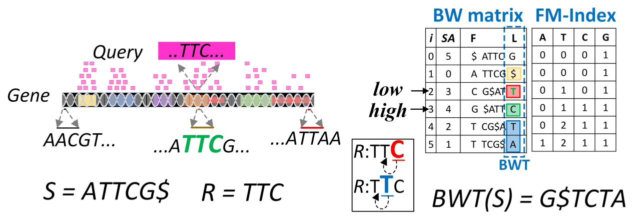
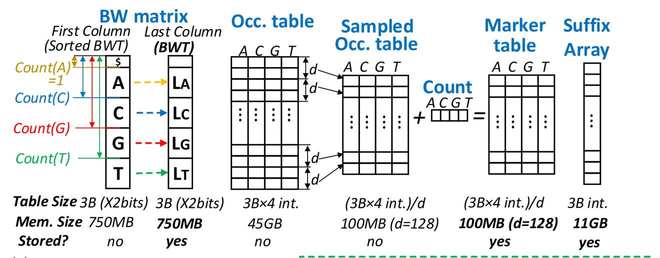


Figure 4: Short read alignment concept.

3 ALIGNS SEQUENCING ALGORITHM

BWT-based Read Mapping: The BWT is a reversible permutation of the characters of a string. Short read alignment algorithms (e.g., BWA [3] 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. 4 gives an intuitive example of such alignment of a sample read-$R = TTC$ to a sample reference $S = ATTCG\$$ 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 (SA) 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. BWT of such reference-S is given by the last column in the BW matrix, here, $BWT(S) = G\$TCTA$. The FM-Index is then built on top of BWT providing the occurrence information of each symbol in BWT. The SA 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 (C in Fig. 4). During the backward search, the matched lower bound (low) and upper bound ($high$) in a SA of the S for each nucleotide in R are determined based on FM-Index and count function [3]. Thus, the result of read searching is represented as a SA 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 [19]. Moreover, BWT-based read mapping algorithms can be simply extended to allow mismatches in the read mapping [3].

Alignment-in-Memory Algorithm: The presented DNA exact alignment-in-memory algorithm is based on BWT and FM-Index sequencing algorithm [3], but optimized using *AlignS*'s functions, i.e. *MEM*, *XNOR_Match*, and *IM_ADD*. As the first step of such process, shown in Fig. 5a, 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 (MT), and SA will be stored in the *AlignS* , 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. 5a, $Count(nt)$ 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


BW matrix
First Column (Sorted BWT)
Last Column (BWT)
Count(A) =1
Count(C)
Count(G)
Count(T)
Occ. table
Sampled Occ. table
Count
Marker table
Suffix Array
Table Size 3B (X2bits) 3B (X2bits) 3B×4 int. (3B×4 int.)/d (3B×4 int.)/d 3B int.
Mem. Size 750MB 750MB 45GB 100MB (d=128) 100MB (d=128) 11GB
Stored? no yes no no yes yes
(a)

```
Initialization: $low \leftarrow 0$, $high \leftarrow |S|-1$
for $i \leftarrow |R|-1$ to 0   /* backward search*/
    $low \leftarrow$ Bound$(MT[\lfloor low/d \rfloor], R[i], low)$
    $high \leftarrow$ Bound$(MT[\lfloor high/d \rfloor], R[i], high)$
    if $low > high$
        break & return $\emptyset$
    end if
end for
```

```
Procedure: Bound(MT, nt, id) /* compute matched interval*/
    count_match ← 0
    for j ← 0 to j < (id mod d)
        if XNOR_Match (nt, BWT[id-(id mod d)+j]) ==1
            count_match = count_match+1
        end if
    end for
    marker ← MEM(MT[⌊id/d⌋, nt])   /*Read MT*/
  return IM-ADD(marker, count_match)
end Procedure
```

(b)

Figure 5: (a) Pre-computation needed in *AlignS*'s alignment algorithm, (b) The *Bound* procedure implementation.

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 MT is constructed by element-wise addition of Sampled Occ-table with $Count(nt)$, which leads to the same size as Sampled Occ-Table. MT contains the matched position of the nucleotides in BWT in the First Column and helps *AlignS* to efficiently retrieve the values of low and $high$ in each iteration.

As shown in Fig. 5b, the read searching operation is mainly implemented through the proposed $Bound(MT, nt, id)$ procedure performed on BWT, which computes the updated interval bound (either low or $high$) value from MT with bucket width d and input index-id. Such procedure is iteratively used in every step of 'for' loop and *AlignS* is especially designed to handle such computation-intensive load through summing the current 'marker' value with the occurrence counting result of the needed nucleotides between checkpoint position and remaining positions in BWT. To implement the *Bound* procedure totally within memory, we exploit three *AlignS*'s functions, i.e. MEM (memory read), XNOR_Match (XNOR2), and IM_ADD (add), as highlighted in Fig. 5b. *MEM* function is to access data in the saved MT or SA based on the provided index. XNOR_Match is to conduct parallel in-memory XNOR logic to determine if current input-nt matches with BWT elements stored in the whole word-line in only one computational cycle. 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 CPU or other computing units.

4 ALIGNS HARDWARE MAPPING

Correlated Data Partitioning: Due to large memory space requirement of pre-computed tables (BWT, MT, and SA) for alignment-in-memory algorithm, we have to partition these tables to fully leverage *AlignS*'s parallelism, and to maximize alignment computation throughput. Given a BWT index range, the accessed memory region of MT and BWT could be easily predicted and computation could be localized if we could store such correlated region into the same memory sub-array. Thus, we propose a novel, correlated data partitioning and mapping methodology as shown in Fig. 6

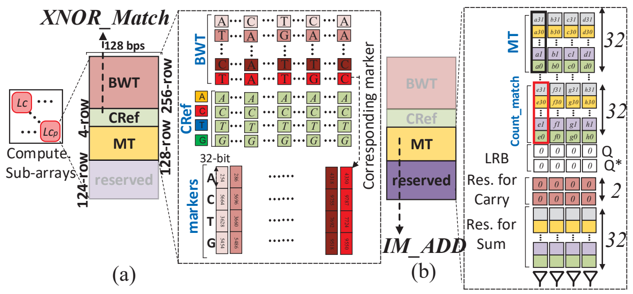


Figure 6: *AlignS*'s sub-array partitioning for efficient local (a) Parallel search and (b) Rank computation.

to locally store correlated regions of BWT and MT vectors in the same memory sub-array and enable fully local computation (i.e. XNOR_Match and IM_ADD completely within the same sub-array without inter-bank/chip communication).

As discussed earlier, *AlignS* architecture consists of multiple memory chips, each consisting of banks, mats, and memory sub-arrays in a hierarchical way. Each sub-array (512 rows×256 columns) is divided into four zones to store four different data types, i.e. BWT, CRef, MT, and reserved space for IM_ADD and buffer (Fig. 6a). First 256 rows are occupied with the corresponding BWT, where each row stores up to 128 bps (encoded by 2 bits). In addition, we propose to store four nucleotide computational reference vectors ($CRef$), in which each vector represents one type of nucleotide with vector size of number of bits in one word-line. CRef is especially designed to enable fully parallel match operation- XNOR_Match. Next to it, the value of markers (MT) is pre-calculated and checkpointed every d (=128) positions (one row), and vertically saved to keep the size in check within *AlignS* platform. Hence, 256 columns are allocated for storing MT, each storing 4-byte value for bps (128-bit). The same colors are used in Fig. 6a to show the BWT rows and the corresponding marker columns. After mapping the data, starting from the rightmost symbol in R (e.g., C in Fig. 4), two steps need to be taken in order to implement *Bound* procedure and return low and $high$ for next symbol-T.

Parallel Search: Considering current input nucleotide is T and input index as 'id', *AlignS*'s Ctrl can readily convert this BWT index into the corresponding memory WL and BL addresses storing data BWT[id-(id mod d)] to BWT[id]. Then, such bits and corresponding CRef-T is sensed at the same time using the *AlignS*'s XNOR circuits to implement the parallel search operations (XNOR_Match). If the XNOR output is '1', representing a match is found, Ctrl's embeded counter counts up to eventually compute 'count_match' for next

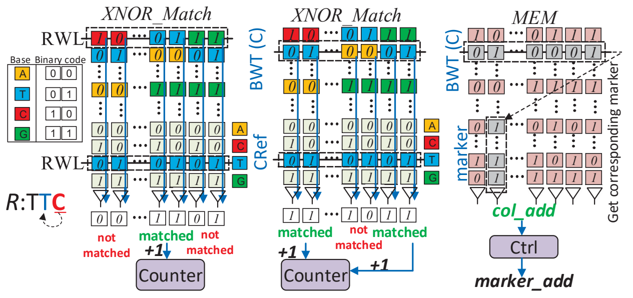
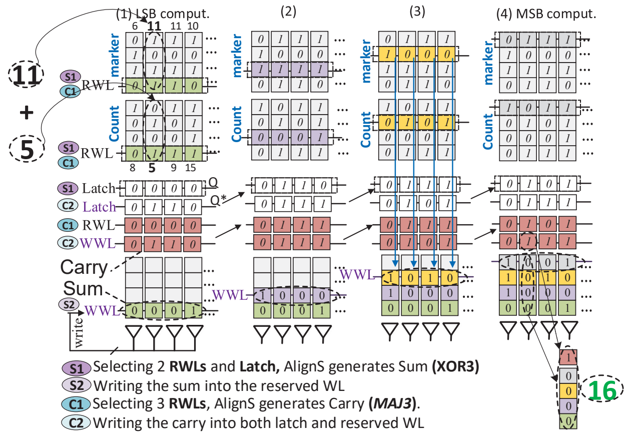


Figure 7: *AlignS*'s parallel search operation.

operation. Fig. 7 shows the XNOR_Match procedure to locate Ts in a sub-array. After parallel alignment of the first row, the Ctrl starts aligning the second row and so on. When counting is done, the sub-array returns the 'count_match' and marker address (marker_add). Note that, the correlated data partitioning methodology guarantees the read of 'marker' value (MEM) is always a local memory access within the same memory array.

Rank Calculation: After parallel search, the 'marker' and just computed and transposed 'count_match' are buffered in MT and reserved memory spaces, respectively, as shown in Fig. 6b, to further implement IM_ADD function. Now, considering n-activated sub-arrays with the reserved row size of 124×256, each sub-array can easily handle the parallel *add* of up to 256 elements of 32-bit to maximize the throughput. The memory sub-array organization for such parallel computation is delineated in Fig. 6b. Two reserved rows for Carry results initialized by zero and 32 reserved rows are considered for Sum results. We have shown the current state (Q) as well as the next state (Q*) of LRB after being enabled for further clarification. Here, we use the *add* operation of two 4-bit values (11 and 5) in Fig. 8 to elaborate how multi-bit addition operates in the *AlignS*. Every two corresponding elements that are going to be added together are aligned in the same bit-line. Now parallel addition can be performed based on the steps detailed in Fig. 8 to generate 16 as the output.



(1) LSB comput.
(2)
(3)
(4) MSB comput.
marker
Count
11
+
5
RWL
Latch
Q
Q*
WWL
Carry
Sum
write
16
S1 Selecting 2 RWLs and Latch, AlignS generates Sum (XOR3)
S2 Writing the sum into the reserved WL
C1 Selecting 3 RWLs, AlignS generates Carry (MAJ3).
C2 Writing the carry into both latch and reserved WL

Figure 8: *AlignS*'s in-memory multi-bit addition used in rank calculation.

Extend to Inexact Alignment: Our discussed alignment algorithm and its mapping to *AlignS* could be easily extended to handle inexact match during short read alignment based on the method fully-explained in [3]. However, due to lack of space, we leave it for the future work. Inexact match for sequence alignment has a tolerance for number of mismatches between short read-R and reference genome-S. We can handle mismatch by recursively calculating the intervals that match $R[0, i]$ with no more than z differences on the condition that $R[i + 1]$ matches $\{low, high\}$. As long as there is still tolerance for differences up to current position i, we consider all possible alignments when updating the intervals. The intervals for position i should take union for all intervals including intervals for match and mismatch. At the end, we report all the target positions in the reference genome that the short read can be mapped to with no more than z mismatches. Such algorithm iteratively uses the *Bound* procedure and can be readily accelerated by *AlignS*.

5 EXPERIMENTAL RESULTS

5.1 Accelerators' setup

To assess the performance of *AlignS* as a new PIM platform, a comprehensive device-to-architecture evaluation framework along with two in-house simulators are developed. First, at the device level, we jointly use the Non-Equilibrium Green's Function (NEGF) and Landau-Lifshitz-Gilbert (LLG) with spin Hall effect equations to model SOT-MRAM bit-cell [13, 15]. For the circuit level simulation, a Verilog-A model of 2T1R SOT-MRAM device is developed to co-simulate with the interface CMOS circuits in Cadence Spectre and SPICE. 45nm North Carolina State University (NCSU) Product Development Kit (PDK) library [20] is used in SPICE to verify the proposed design and acquire the performance. Second, an architectural-level simulator is built based on NVSim [21]. Based on the device/circuit level results, our simulator can alter the configuration files (.cfg) corresponding to different array organization and report performance metrics for PIM operations. The controllers and add-on circuits are synthesized by Design Compiler [22] with an industry library. Third, a behavioral-level simulator is developed in Matlab taking architectural-level results to calculate the latency and energy that *AlignS* spends on alignment task based on alignment-in-memory algorithm. It has a mapping optimization framework to maximize the performance w.r.t. the available resources.

Accelerators: We perform an extensive comparison with state-of-the-art accelerators including SW-based Darwin [1], RaceLogic [6], ReCAM [23], and FM-Index-based acceleration solutions such as Soap2 [5]/Soap3-dp [5] on CPU/GPU, AligneR [2], FPGA [8], and ASIC [7]. Due to the lack of space, we refer the readership to the papers for the detailed configuration of each accelerator. Note that, to perform short read alignment on CPU/GPU, we use Soap2/Soap3 [5] considering only reads with $\leq$2 mismatches. Besides, ReRAM-based arrays and CAMs are simulated with NVSim [21] and NVSim-CAM [24], respectively. **Methodology:** To evaluate the performance of *AlignS* and other accelerators and provide a solid comparison, we generate 10 million 100-bp short read queries via ART simulator [25] and align them to the human genome Hg19 [1].

5.2 Results

Figure 9a and b report the power consumption and throughput of under-test accelerators, respectively.

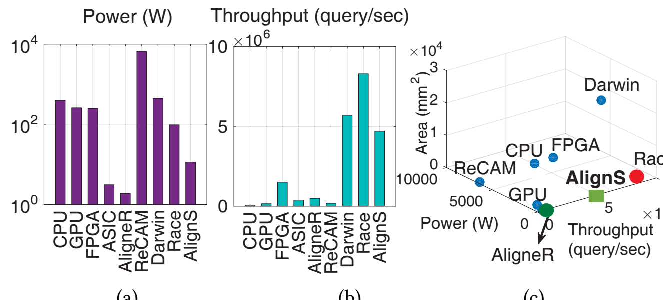


Figure 9: (a) Normalized log scale power consumption, (b) Throughput, (c) Trade-off between area, power, and throughput of different accelerators compared to *AlignS*.

Power-Throughput-Area Trade-offs: We observe that ASIC design [7] and ReRAM-based AligneR [2] consume the least power

[1]The population variation and genome error rate were set to 0.1% and 0.2%, respectively.

compared to other designs, where *AlignS* stands as the third-best power-efficient design. From the throughput point of view, Race Logic [6] and Darwin [1] show the best performance compared to others. However, *AlignS* can show the highest throughput compared to others such as GPU, ASIC, FPGA, ReCAM, and AligneR due to its massively-parallel and local computational scheme. Therefore, we can analyze the existing trade-offs between power, throughput, and area as Fig. 9c. 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. $Throughput/Watt$. *AlignS* can improve 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 [2], ASIC [7], and FPGA [8] acceleration solutions, respectively. Table 1 also reports throughput per Watt per mm^2 for different accelerators. Considering the area factor, we observe that *AlignS* 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*'s parallel computing schemes can be leveraged to accelerate short read alignment and provide ultra-high internal bandwidth.

Table 1: Performance of short read alignment accelerators.

	GPU	ASIC	FPGA	AligneR	Darwin	Race	ReCAM	AlignS
Throughput/Watt	581.3	122K	6.1K	259.6K	12.8K	85K	26.81	**412.28K**
Throughput/Watt/mm^2	0.39	547	0.42	7.2K	0.47	47	0.24	**6.6K**

Memory Wall: Figure 10a shows the required off-chip memory access for different accelerators. We observe that all the FM-Index accelerators including CPU/GPU[5], ASIC [7], and FPGA [8], except PIM platforms (i.e. AligneR, ReCAM, and *AlignS*) heavily rely on off-chip memory consuming humongous energy to fetch data from stored tables and queries. Note that, ASIC design performs FM-Index-based alignment with 1GB off-chip memory after compression. Figure 10b reports the Memory Bottleneck Ratio (MBR), i.e. the time fraction at which the computation has to wait for data and on-/off-chip data transfer obstructs its performance (memory wall happens). The evaluation is performed according to the peak throughput for each platform considering number of memory access. The results show the *AlignS*'s efficiency for solving memory wall issue. We observe that *AlignS* spends less than ~15% time for memory access and data transfer owning to the PIM acceleration schemes. Note that, ASIC and other PIM platforms spend less than 50% time waiting for the loading data. AligneR solution shows higher memory bottleneck ratio compared with *AlignS* due to its unbalanced computation and data movement. The less MBR can be translated as the higher Resource Utilization Ratio (RUR) for the accelerators plotted in Fig. 10c. We observe that *AlignS* has the highest resource utilization with up to ~76%. Overall, PIM solutions demonstrate high ratio (>60%) which reconfirms the results reported in Fig. 10b.

6 CONCLUSION

In this paper, we propose an efficient processing-in-memory accelerator based on SOT-MRAM (*AlignS*) to execute short read alignment based on a hardware-friendly alignment algorithm. *AlignS* is optimized through a new correlated data partitioning and mapping technique that provides local storage and processing of indices to fully exploit the algorithm-level's parallelism to accelerate both exact and inexact matches. The results show that *AlignS* improves the read alignment throughput per Watt per mm^2 by ~12× compared with the ASIC design. Besides, it achieves 1.6× higher throughput per Watt compared to recent FM-index-based ReRAM platform.

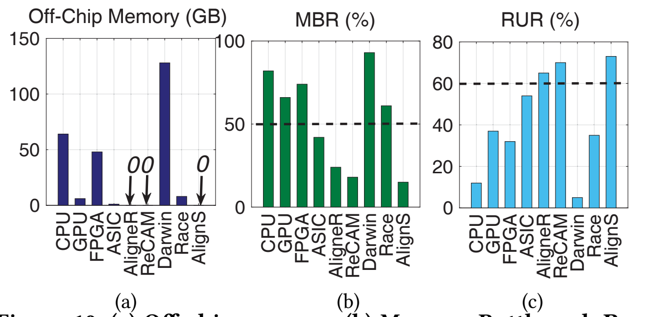


Figure 10: (a) Off-chip memory, (b) Memory Bottleneck Ratio, (c) Resource Utilization Ratio for different accelerators.

ACKNOWLEDGEMENTS

This work is supported in part by the National Science Foundation under Grant No. 1740126, No. 1755761, and Semiconductor Research Corporation nCORE.

REFERENCES

[1] 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.
[2] F. Zokaee *et al.*, "Aligner: A process-in-memory architecture for short read alignment in rerams," *IEEE Computer Architecture Letters*, 2018.
[3] H. Li and R. Durbin, "Fast and accurate short read alignment with burrows–wheeler transform," *bioinformatics*, vol. 25, pp. 1754–1760, 2009.
[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.-C. Wu *et al.*, "A 135-mw fully integrated data processor for next-generation sequencing," *IEEE TBioCAS*, vol. 11, pp. 1216–1225, 2017.
[8] J. Arram *et al.*, "Leveraging fpgas for accelerating short read alignment," *IEEE/ACM TCBB*, vol. 14, pp. 668–677, 2017.
[9] V. Seshadri *et al.*, "Ambit: In-memory accelerator for bulk bitwise operations using commodity dram technology," in *Micro*. ACM, 2017, pp. 273–287.
[10] W. Huangfu *et al.*, "Radar: a 3d-reram based dna alignment accelerator architecture," in *55th DAC*. ACM, 2018, p. 59.
[11] S. K. Khatamifard *et al.*, "A non-volatile near-memory read mapping accelerator," *arXiv preprint arXiv:1709.02381*, 2017.
[12] L. Yavits *et al.*, "Resistive associative processor," *IEEE Computer Architecture Letters*, vol. 14, pp. 148–151, 2015.
[13] S. Angizi *et al.*, "Cmp-pim: an energy-efficient comparator-based processing-in-memory neural network accelerator," in *55th DAC*. ACM, 2018, p. 105.
[14] S. Angizi, Z. He, and D. Fan, "Dima: a depthwise cnn in-memory accelerator," in *2018 ICCAD*. IEEE, 2018, pp. 1–8.
[15] X. Fong *et al.*, "Spin-transfer torque devices for logic and memory: Prospects and perspectives," *IEEE TCAD*, vol. 35, 2016.
[16] S. Li *et al.*, "Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories," in *2016 53nd DAC*. IEEE, 2016.
[17] S. V. Kosonocky *et al.*, "Enchanced multi-threshold (mtcmos) circuits using variable well bias," in *ISLPED*, 2001, pp. 165–169.
[18] H. Ozdemir *et al.*, "A capacitive threshold-logic gate," *IEEE Journal of Solid-State Circuits*, vol. 31, no. 8, pp. 1141–1150, 1996.
[19] S. Canzar *et al.*, "Short read mapping: An algorithmic tour," *Proceedings of the IEEE*, pp. 436–458, 2017.
[20] (2011) Ncsu eda freepdk45. [Online]. Available: http://www.eda.ncsu.edu/wiki/FreePDK45:Contents
[21] X. Dong *et al.*, "Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *IEEE TCAD*, vol. 31, 2012.
[22] P. Kurup *et al.*, *Logic synthesis using Synopsys®*, 2012.
[23] R. Kaplan *et al.*, "A resistive cam processing-in-storage architecture for dna sequence alignment," *IEEE Micro*, vol. 37, pp. 20–28, 2017.
[24] S. Li *et al.*, "Nvsim-cam: a circuit-level simulator for emerging nonvolatile memory based content-addressable memory," in *35th ICCAD*. ACM, 2016.
[25] W. Huang *et al.*, "Art: a next-generation sequencing read simulator," *Bioinformatics*, vol. 28, pp. 593–594, 2011.