RePAIR: A ReRAM-based Processing-in-Memory Accelerator for Indel Realignment

Ting Wu[*†], Chin-Fu Nien[†], Kuang-Chao Chou[‡], Hsiang-Yun Cheng[†]
[*]Electrical and Computer Engineering, Carnegie Mellon University, USA
[†]Research Center for Information Technology Innovation, Academia Sinica, Taiwan
[‡]Gradute Institute of Electronics Engineering, National Taiwan University, Taiwan
Email: [*]tingwu@andrew.cmu.edu, [†]{watchmannien, hycheng}@citi.sinica.edu.tw, [‡]r10943002@ntu.edu.tw

***Abstract*—Genomic analysis has attracted a lot of interest recently since it is the key to realizing precision medicine for diseases such as cancer. Among all the genomic analysis pipeline stages, Indel Realignment is the most time-consuming and induces intensive data movements. Thus, we propose RePAIR, the first ReRAM-based processing-in-memory accelerator targeting the Indel Realignment algorithm. To further increase the computation parallelism, we design several mapping and scheduling optimization schemes. RePAIR achieves 7443× speedup and is 27211× more energy efficient over the GATK3.8 running on a CPU server, significantly outperforming the state-of-the-art.**

I. Introduction

Genomic analysis has received intensive interest recently, as it plays a vital role in realizing precision medicine to provide an accurate diagnosis for diseases. A typical genomic analysis flow processes a massive amount of data generated by genome sequencing machines. It goes through several time-consuming stages to detect the genomic variants, such as genetic mutations associated with cancer [1]. Accelerating genomic analysis can help to provide timely treatment for patients and lower the fatality rate. While most prior studies focus on accelerating the algorithms adopted in the primary alignment stage of genomic analysis [2]–[8], the Indel Realignment stage, which aims to correct the errors generated by the primary alignment stage, actually takes a longer execution time (34% of total execution time, 6× slower than primary alignment algorithms) [9]. Indel Realignment is crucial, especially for cancer diagnosis, as the results of primary alignment contain 0.5%-2% errors while the variants exist only at a frequency of $1/10^{-5}$ [9]. There is a lack of accelerator designs targeted at speeding up such a critical stage.

One promising solution to increase computation parallelism and mitigate the energy overheads induced by massive genome data movements for Indel Realignment is designing a processing-in-memory (PIM) accelerator. When performing the Indel Realignment algorithm, two kinds of operations, comparison and weighted Hamming distance (*whd*) calculation, are mainly required, while *whd* can be modeled as matrix-vector-multiplication (MVM). Prior studies show that metal-oxide resistive random access memory (ReRAM) enables in-situ computations directly in the memory for both kinds of operations [5], [10]–[15]. Thus, we propose RePAIR, an energy-efficient ReRAM-based PIM design for Indel Realignment.

To accelerate Indel Realignment via ReRAM-based PIM architecture, several design challenges need to be addressed. First, due to the huge size of genome datasets, a large amount of *whd* needs to be calculated It would be beneficial to compute more data simultaneously to enhance parallelism without additional hardware requirements. Second, since the length of genomic data varies from time to time, the scheduling method applied will greatly affect performance. We propose several optimization techniques to tackle these design challenges. The main contributions of this work can be summarized as follows:

- We propose RePAIR, a ReRAM-based accelerator for Indel Realignment. To the best of our knowledge, we are the first work that effectively exploits the high degree of parallelism and minimal data movement overheads of the PIM architecture to enable energy-efficient Indel Realignment.
- We propose a mapping scheme to increase the parallelism degree when calculating *whds* while lowering the required hardware resources such as the analog-to-digital converters.
- We propose a novel way to schedule the *whds* computations. To perform the comparison operations at the largest parallelism degree, we duplicate either the consensus sequences or the reads. The throughput per cycle is highly improved.
- The proposed RePAIR architecture achieves 7443× speedup and is 27211× more energy efficient over the GATK3.8 benchmark running on an 8-thread CPU server, outperforming the state-of-the-art FPGA-based acceleration.

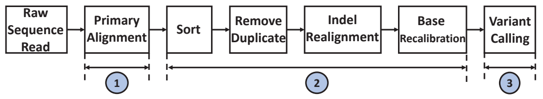


Fig. 1. A typical genomic analysis flow which contains three stages: ① primary alignment, ② alignment refinement, and ③ variant calling.

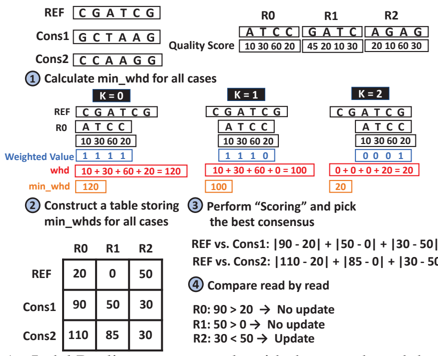
II. Background

A. Indel Realignment

Genomic analysis aims to identify nucleotide differences between an individual genome and the reference genome. Its typical flow [16] is composed of three major pipelines, as shown in Fig. 1. Indel Realignment is part of the alignment refinement pipeline, and its purpose is to correct the structural errors generated during primary alignment to recover the original gene sequence. Specifically, reads (i.e., DNA subsequences) with insertions/deletions (indels) at its end are sometimes aligned incorrectly during primary alignment. Indel Realignment identifies likely indels observed across all reads. It realigns reads against those targets on reference and consensus sequences, where a consensus sequence is a guess of the original gene sequence. Although the newly released GATK4 does not use this stage, it is vital for the cases with low-frequency somatic variants that are difficult to detect.

Algorithm. The Indel Realignment algorithm contains three steps. First, for a given target interval, it calculates the minimum *whd* (*min_whd*) for each read-consensus and read-reference pair. Then, it performs the "scoring" of each consensus and selects the one with the minimum score. Finally, based on the scores, reads are realigned if necessary.

Example. Fig. 2 elaborates this algorithm with a simple case. First, we calculate the *min_whd* for each read against each consensus (including the reference). We take one of the cases that calculates the *min_whd* for the reference-R0 pair as an example. At index *k=0*, we compare the nucleotide bases from read R0 (ATCC) to the corresponding reference (CGAT). If the base pair matches, the corresponding read quality score is multiplied by zero (i.e., weight is zero). Otherwise, it is multiplied by one. Then, we sum the four weighted read quality scores to get the *whd* at this index. Afterward, we right-shift the read by one base and perform the same process again to get the *whd* for *k=1*. The smaller one between the *whd* of *k=0* and *k=1* is recorded as the *min_whd*. This process continues until the rightmost base of the read corresponds with the counterpart of the reference. We record the *min_whd* of each consensus-read pair in a table, as shown in Fig. 2-②. Based on the table, we calculate the score of each consensus by summing the absolute differences between the *min_whd* of each consensus and the reference for all reads. Then, the one with the minimum score is picked as the best consensus. Finally, we compare the *min_whd* of the best consensus against the *min_whd* of the reference read-by-read and update the start positions for all reads where the consensus scored better.



REF C G A T C G
Cons1 G C T A A G
Cons2 C C A A G G
R0 R1 R2
A T C C G A T C A G A G
Quality Score 10 30 60 20 45 20 10 30 20 10 60 30
① Calculate min_whd for all cases
K = 0
K = 1
K = 2
Weighted Value 1 1 1 1
whd 10 + 30 + 60 + 20 = 120
min_whd 120
1 1 1 0
10 + 30 + 60 + 0 = 100
100
0 0 0 1
0 + 0 + 0 + 20 = 20
20
② Construct a table storing min_whds for all cases
③ Perform "Scoring" and pick the best consensus
REF vs. Cons1: |90 - 20| + |50 - 0| + |30 - 50| = 140 √
REF vs. Cons2: |110 - 20| + |85 - 0| + |30 - 50| = 195
④ Compare read by read
R0: 90 > 20 → No update
R1: 50 > 0 → No update
R2: 30 < 50 → Update

	R0	R1	R2
REF	20	0	50
Cons1	90	50	30
Cons2	110	85	30

Fig. 2. An Indel Realignment example with three reads and three consensuses. Cons1 is chosen as the best consensus, and only R2 is updated.

B. ReRAM Basics

ReRAM is a type of non-volatile memory that can provide high density with low leakage. A typical ReRAM cell has an oxide layer (e.g., HfO_x) sandwiched between two metal layers of electrodes [17], as shown in Fig. 3(a). Applying voltage pulses across a ReRAM cell can switch to a high resistance state (HRS) or low resistance state (LRS). Initially, a FORM operation is required to create a conductive filament (CF) to connect the two electrodes layers. Then, the CF is either ruptured (RESET from LRS to HRS) or recovered (SET from HRS to LRS) depending on the voltage polarity, and it can be eliminated via a BREAK operation. ReRAM writes can be implemented by bipolar switching or unipolar switching, while

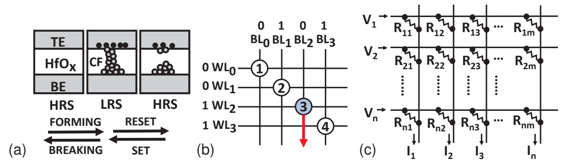


Fig. 3. (a) A typical ReRAM cell structure; Implementation of (b) comparison using bipolar ReRAM cells and (c) MVM operation on a crossbar array.

bipolar switching is better for endurance [18] owing to less material loss. In a ReRAM crossbar array, each cell is located at the cross point of a wordline (WL) and a bitline (BL). The bipolar switching SETs the cell with positive voltage (defined as high WL and low BL voltage) and RESETs the cell with a negative voltage (defined as low WL and high BL voltage).

C. ReRAM-based Computations

ReRAM crossbar arrays can be utilized to perform comparison operations [5] and matrix-vector multiplications (MVMs) [10]–[15] in one constant time step. When implementing the Indel Realignment algorithm, we need to compare nucleotide bases and sum up the weighted read quality scores to get the *min_whd* for each consensus-read pair. Thus, it is promising to leverage the computing-in-memory capability of ReRAM to accelerate Indel Realignment.

Comparison operation. Instead of using unipolar switching ReRAM to implement comparison operations as in prior work [5], bipolar switching ReRAM is utilized in this paper to enhance endurance [18]. Here we demonstrate that it is possible to implement comparison operations on a ReRAM crossbar array with bipolar cells through a proper encoding scheme. Fig. 3(b) illustrates an example. We first FORM the diagonal cells. Other cells are not FORMed, so they cannot be SET or RESET. Before performing each comparison, we RESET all the diagonal cells. Then, we apply low voltage (0) or high voltage (1) on WLs and BLs to represent the binary encoding of the two values to be compared. Since a bipolar cell switches to LRS only when a positive voltage is applied, only cell ③ in Fig. 3(b) is SET. As such, when we apply a sensing voltage to all the WLs while all the BLs are grounded (i.e., READ the crossbar array), only 1× LRS sensing current is read out by the sense amplifier (SA) from the crossbar array. Note that one-hot encoding is used to represent the values to be compared, such that no current will be read out once the two values are matched, while 1× LRS sensing current will be detected if the two values are mismatched. Implementing comparison operations may damage cells in a short period as cells are frequently written. We can switch the working cells (FORMed cells) among different diagonal lines once in a while and add error correction pointers to enable reliable computation without introducing errors for years [5].

Matrix-vector multiplication. Fig. 3(c) illustrates how to efficiently perform MVM using a ReRAM crossbar array. If we apply READ voltages V_i to each WL and ground each BL, the accumulated current on bitline j is equal to the sum-of-products of the input voltage and cell conductance ($I_j = \sum_i V_i/R_{ij} = \sum_i V_i \times G_{ij}$). Analog-to-digital converters (ADCs) are required to convert the output current on each bitline to digital values.

III. RePAIR Architecture and Design

A. Architecture

A RePAIR chip comprises multiple tiles connected with an on-chip mesh interconnect. Fig. 4 shows the architecture of a tile. There are four Computation Units (CUs), one eDRAM buffer, and several simple arithmetic logic units (sALU) within a tile, all connected with a shared bus. Each CU consists of several ReRAM crossbar arrays, sensing circuits (SAs and ADCs), shift-and-adds (S+As), an input register (IR), and an output register (OR). The ReRAM crossbar arrays within a CU are divided into two groups: comparison (Comp) array and MVM array. Comp arrays are responsible for comparing the nucleotide bases of consensuses (including reference) and reads, while MVM arrays implement the calculation of *whd*.

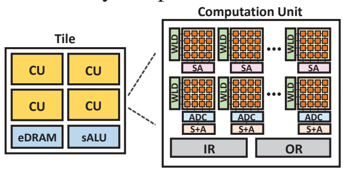


Fig. 4. RePAIR architecture for Indel Realignment.

B. Implementation of Indel Realignment Algorithm

Before each tile performs Indel Realignment for a target interval, the consensuses (including reference) and reads to be compared are first loaded into the eDRAM buffer and then distributed to the IR of each CU. Meanwhile, the quality scores of reads are mapped and programmed into the MVM arrays. The first step of the algorithm is to compare the nucleotide bases of consensuses and reads, so these sequences are concatenated and loaded into the Comp array as inputs to the WLs and BLs, as shown in Fig. 5. All the reads are compared with the reference and consensuses at the same time, and the compared results are then fed into the MVM arrays to calculate the *whds*. The output *whds* are written to the OR and then stored in the eDRAM buffer. We then right-shift the position of the consensus sequences, as shown in Fig. 5, and perform the comparison and MVM again to get the *whds* of the next index position. After each index shifting, the calculated *whds* are compared with the *min_whds* using sALU, and the smaller one and its corresponding index are recorded in the eDRAM buffer. This process continues until each read is compared with all the references and consensus. Finally, we use the sALU to perform the "scoring" process and compare the *min_whd* of the best consensus against the reference read-by-read to update the start positions for all reads where the consensus scored better. Our design enables highly-parallel computation, as the comparison and *whd* calculation for different consensus-read pairs can be performed simultaneously.

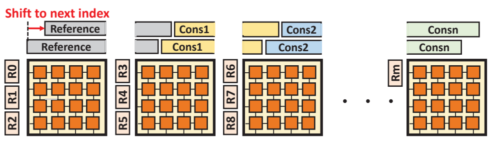


Fig. 5. The schematic diagram of how we arrange the position of reads and consensuses on Comp arrays. The consensuses and reads are concatenated and distributed across different Comp arrays to increase computation parallelism.

Nucleotide bases comparison. Fig. 6(a) shows an example to demonstrate how to use the Comp array to compare nucleotide bases of reads and consensuses. We use one-hot encoding to represent four nucleotide bases: *A=0001*, *T=0010*, *C=0100*, *G=1000*. Initially, the main diagonal cells (①-⑧) of the 8×8 array are FORMed, with the rest of the cells left unFORMed. The comparison operation is implemented by three pipeline stages: RESET, COMP, and READ. At the RESET stage, the diagonal cells are all reset to HRS. Then, at the COMP stage, we apply 1.8V and 0V, indicating 1 and 0, to the WLs and BLs based on the one-hot encoding of the input nucleotide bases. In this example, we apply 01000001 (CA) on BLs and 01001000 (CG) on WLs. Cells ①-⑧ switch to LRS if a positive voltage is applied, otherwise stay in HRS. Since one-hot encoding is employed, only one cell among every four diagonal cells would switch to LRS if the two bases are mismatched. In this example, A and G are mismatched. Only cell ⑤ is switched while the other diagonal cells remain in HRS. Finally, at the READ stage, we apply the sensing voltage to all WLs, ground all BLs, and use SAs to read out the sensing current from the diagonal cells. The outputs from SAs represent the weighted values and will later be the inputs of MVM arrays.

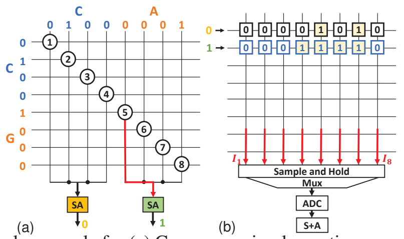


Fig. 6. A simple example for (a) Comp array implementing compare operation and (b) MVM array implementing *whd* calculation. The outputs of SAs are applied to WLs in MVM arrays.

Calculation of *whd*. Fig. 6(b) uses the same example to demonstrate how to use MVM arrays to calculate *whd*. Assume that the read quality score of nucleotide C is ten and that of G is thirty, 00001010 and 00011110 are mapped to the cells on the first and second WLs if the bit-precision of each quality score is 8-bit and single-level cells are used. The weighted values read out from the SAs of the Comp Array are applied as inputs to the WLs. Then, the accumulated current (sum-of-products result) on each BL is held by sample-and-hold circuits and converted into digital values via the shared ADC. After using S+A to assemble the results from each BL, we get the *whd* (30 in this example) of the consensus-read pair.

Mapping of read quality scores. To fully exploit the parallel computation capability of ReRAM crossbar arrays, the mapping of read quality scores on MVM arrays must support parallel *whd* computations for multiple consensus-read pairs. Assume that each read has 36 base pairs (36 bps), and each quality score is represented by 8 bits, Fig 7(a) shows the mapping of quality scores from a single read. To enable parallel *whd* computation, we map scores of other reads on the same MVM array diagonally, as shown in Fig. 7(b). The degree of parallelism (7 in this example) depends on the read length and the size of the MVM array. To avoid wasting the rest of the unused space in the MVM array, we follow the same way to place read quality scores of other target intervals,

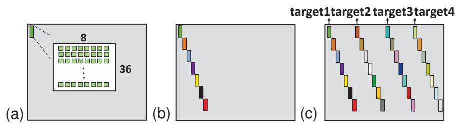


Fig. 7. The mapping of 36bps reads on a 256×256 MVM array with single-level ReRAM cells: (a) mapping of a single read; (b) mapping of multiple reads in the same target interval for parallel computation; (c) mapping of quality scores from different target intervals for better space utilization.

which we will process after the computations of the current target interval are completed, as shown in Fig. 7(c).

C. Mapping and Scheduling Optimization

Asynchronous target scheduling. The length of target intervals differs a lot, while the number of CUs and crossbars per tile is fixed. For the target intervals with a small amount of data, we can merely use part of the CUs to perform the computation. At the same time, the rest of the CUs will be idle until the slowest CU in the same tile completes. To fully utilize the available CUs and improve performance, we allow the next target to be launched as soon as a CU is free.

Sequence duplication for highly-parallel comparison. The length of cascaded consensuses and cascaded reads may mismatch. As such, using the *naïve* scheme shown in Fig. 8(a) to perform comparison operations may underutilize the available parallel computing capability provided by ReRAM crossbar arrays. In this example, the cascaded consensus length is far shorter than the cascaded read length. Thus, many Comp arrays are idle, and we need to shift the cascaded consensuses a lot of times to finish the comparisons for this target interval. Here, we come out with a *proposed* scheme to significantly speed up this part. We duplicate the cascaded consensuses, as shown in Fig. 8(b). In this way, we can largely reduce the number of idle Comp arrays and the required number of shifts.

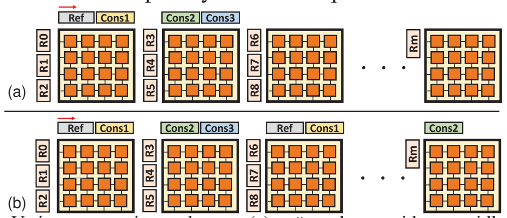


Fig. 8. Various comparison schemes: (a) *naïve* scheme with many idle Comp arrays; (b) duplicate consensus sequences to increase crossbar utilization.

Grouped comparison in a pipelined manner. The Comp array and MVM array take a different number of cycles to complete one set of *whd* computation. Thus, we propose a grouped mapping and scheduling scheme to tackle the challenges introduced by this mismatch in processing speed. A Comp array takes three cycles to perform RESET, COMP, and READ stages, while an MVM array only takes one cycle for MVM computation. Fig. 9 shows a simple example. Suppose all the Comp arrays start the computation simultaneously, as shown in Fig. 10(a). In that case, the MVM arrays in Fig. 9(a) will idle for three cycles until the Comp arrays obtain the weighted values in the READ stage and then concurrently perform MVM operations. Assume it takes 5 ADCs to complete the analog-to-digital conversion for the computation with 7 quality scores in a single 256×256 MVM array within a cycle, 15 ADCs in total are required. The

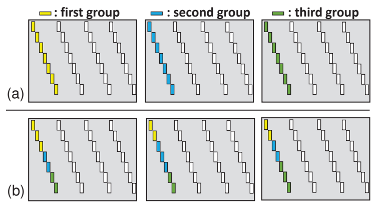


Fig. 9. A simple example with three MVM arrays in one CU: (a) each MVM array idles for 3 cycles then simultaneously performs MVM operations for 7 quality scores in a single cycle; (b) the proposed mapping for grouped Comp array, where each MVM array does not idle and performs MVM operations for at most 3 quality scores in a single cycle.

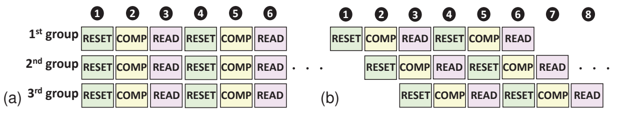


Fig. 10. Example computation flow of Comp arrays: (a) all Comp arrays operate simultaneously; (b) Comp arrays are divided into three groups and operated in a pipelined manner.

large amount of ADCs and activated WLs and BLs lead to a significant power and area overhead. Hence, we divide the Comp arrays into three groups operated in a pipelined manner and map the read quality scores in another way to lower the power and area consumption while maintaining the computation throughput. As shown in Fig. 10(b), the Comp arrays are divided into three groups, and different groups are operated in a pipelined manner to maintain similar throughput per cycle. The quality scores of each group are distributed and mapped to different MVM arrays, as shown in Fig. 9(b), so that each MVM array needs to compute at most three quality scores in a cycle. Thus, fewer ADCs are required, resulting in less power and area overheads.

IV. Evaluation and Results

A. Evaluation Setup

We implement a Python-based in-house cycle-accurate simulator to evaluate the computation time and energy consumption of the ReRAM-based accelerator. Table I shows the hardware configuration for the proposed architecture. We use CACTI [19] to model the latency and energy for all buffers along with peripheral circuits such as drivers and SAs at 45nm technology. ReRAM cell model from [17] is adopted, and adder technology required by S+As and sALUs is derived from [20]. We use 45nm 8-bit ADCs in our design, and the power and latency of the ADC are scaled from [21].

As a comparison baseline, we run the Indel Realignment workload using GATK3.8 [16] on a well-characterized NA12878 genome, which is a human genomic DNA sample from the 1000 Genomes project, as reads aligning to the reference genome HS37D5. Twenty-two chromosomes from HS37D5 are selected as our benchmarks, and each of them has a different target number and length. The computation is carried out by Intel Xeon Gold 6144 CPU, where we limit the number of working CPU threads to only 8 threads on 4 cores according to the setting in the prior work [5]. For an apple to apple comparison, we only measure the time and energy spent within the code sections that our accelerator targets. The energy consumption is estimated based on the computation time, runtime CPU utilization, and Intel specification [22].

TABLE I
HARDWARE CONFIGURATION

Tile configuration (32 tiles in total), $107mm^2$		
Component	**Spec**	**Power (per-component)**
eDRAM buffer	size: 82KB; bus width: 512 bits	25.45mW
sALU	16-bits; number: 128	0.69mW

CU configuration (4 CUs per tile)		
Component	**Spec**	**Power (per-component)**
ReRAM array	size: 256×256; number: 256 (236 Comp and 20 MVM array)	33.54μW (write per cell) 386.7nW (read per cell)
WLD	number: 256×256	1.17μW
SA	number: 236×64	125μW
ADC	8 bits; number: 20×4 frequency: 1.2GSps	2.82mW
S+A	number: 20×4	0.69mW
IR	size: 16KB	28.3mW
OR	size: 8KB	17.1mW

B. *Results and Discussion*

1) Performance and Energy: Fig. 11 shows the speedup and energy consumption of RePAIR over the GATK software benchmark running on the baseline CPU. Two schemes are compared to the GATK baseline. The first scheme is the *naïve* scheme, where the sequence duplication method introduced in Section III-C is not applied. The second scheme is the *proposed* scheme, where the shorter sequence of the cascaded consensuses or cascaded reads is duplicated and appended to achieve higher throughput. Fig. 11(a) shows the performance speedup plotted in log-scale. With the highly parallel ReRAM-based computations, even the *naïve* scheme achieves 3653$\times$ speedup on average. The *proposed* scheme provides 7443$\times$ speedup on average, which is around 2$\times$ better than the *naïve* scheme due to the better utilization of computing resources. These are far better than the state-of-the-art [9], an FPGA-based design that achieves 81.3$\times$ speedup over the baseline running on an eight-thread CPU. The high speedup achieved by our designs can be attributed to the following reasons. First, through our highly-parallel pipelined design, about 17.5 KB data (i.e., *whd* result) can be produced every 10ns cycle. Second, the time spent on data transfers between memories and CPU can be alleviated by computing directly in memory.

Fig. 11(b) shows the energy efficiency compared to the CPU baseline. Compared to the baseline, the *naïve* scheme and the *proposed* scheme consume 21474$\times$ and 27211$\times$ less energy on average respectively. The significant energy savings come from the reduction of data transfers between processors and memory units. The *proposed* scheme consumes less energy than the *naïve* scheme because the *proposed* scheme requires less number of sequence shifts, resulting in fewer eDRAM buffer accesses and less aggregation of partial results.

2) Endurance: Endurance is a critical issue for reliable computing. Though SET and RESET are performed frequently in the Comp arrays, we leverage several techniques to enhance endurance. We read out the cell value before performing RESET and only RESET the cell which is SET in the comparison stage to avoid unnecessary RESET of cells. In addition, we evenly distribute the number of writes across all the ReRAM cells by two wear-leveling techniques. First, we FORM different diagonal cells for computation when the accumulated amount of writes reaches a threshold value, which is set as 100K based on a prior work [5]. Second, since the distribution of each type of base (namely, A, T, C, and G) is uneven, we shift the encoding of each base by one bit after each round of computation. Following a previous work [5], we estimate the lifetime of RePAIR by modeling the cell endurance variation as a truncated normal distribution with $\mu = 10^{10}$ and coefficient of variance $CoV \triangleq \sigma/\mu = 0.25$. Several factors affect the overall lifetime, such as the crossbar size, overall throughput, and the total number of ReRAM cells in Comp arrays. Our evaluation shows that RePAIR can perform Indel Realignment on well-known datasets such as NA12878 for more than 7.6×10^9 times without introducing endurance failure.

3) Sensitivity Study: Fig. 12(a) shows the total energy and latency of the proposed architecture normalized to the CPU baseline when the number of Comp arrays is 2$\times$ to 16$\times$ of MVM arrays. Results show that the highest performance is achieved when the number of Comp arrays is about 12$\times$ of MVM arrays. Since the Comp arrays are divided into three groups (i.e., RESET, COMP, and READ), only one-third of Comp arrays can be in the READ stage and output data in each cycle. Every four BLs on a Comp array forms a result of a base pair, which will be fed to one WL on the MVM array. That is why the latency is optimal when the number of Comp arrays is 12$\times$ of MVM arrays. Fig. 12(a) also shows that the energy consumption generally increases as the proportion of Comp arrays grows. With more Comp arrays, fewer MVM arrays are available to complete a large amount of *whd* computations. As a result, more WLs and BLs are activated per MVM array, leading to higher energy consumption. Another trade-off not shown in the figure is that the number of ADCs in each CU grows as the proportion of Comp arrays decreases, resulting in a larger area overhead. For the ratio 12 optimized for latency, the chip area is $107mm^2$, while the chip area is $227mm^2$ for the ratio 3 optimized for energy. Considering the trade-off among latency, energy, and area, we set the ratio to 12 to deliver maximum throughput with a relatively small die size.

Fig. 12(b) shows the energy and latency of the proposed architecture normalized to the CPU baseline under fixed ReRAM capacity when crossbar size varies. Within the same accelerator capacity, the number of crossbars increases as the crossbar size decreases. With a higher number of available crossbar arrays, more *whd* calculations can be done in a single cycle. As a result, the overall execution latency decreases when crossbar size decreases. Fig. 12(b) also shows the energy consumption increases as crossbar size grows. Since the quality scores are mapped onto MVM arrays in a diagonal pattern, more ReRAM cells are unused when the size of crossbars increases. The unused ReRAM cells still contribute to the total energy as their WLs and BLs are still activated. For example, assuming the length of reads is 36, and each read quality score is represented by 8 bits, four 36×8 crossbar regions are activated when the crossbar size is small while one 144×32 crossbar region is activated when the crossbar size is large. Thus, the energy consumption grows rapidly when crossbar size exceeds the length of a single read quality score. Another impact factor not shown in Fig. 12 but affects

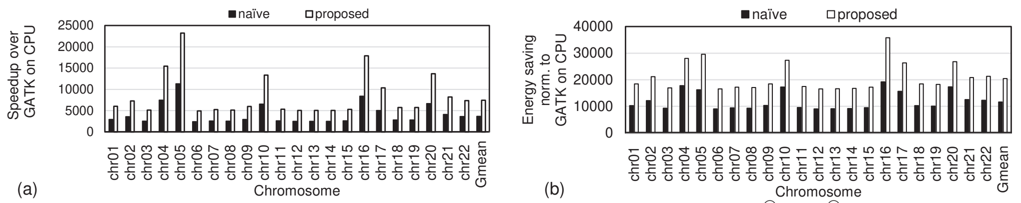


Fig. 11. (a) Speedup and (b) energy efficiency normalized to the GATK result running on Intel® Xeon® Gold 6144 Processor.

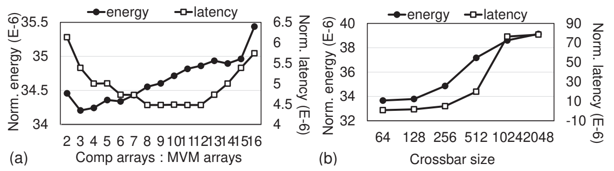


Fig. 12. Energy and latency of RePAIR normalized to CPU when (a) the ratio of Comp arrays to MVM arrays and (b) the crossbar size varies.

the selection of crossbar size is the overhead of peripheral circuits. The number of peripheral circuits grows by twice as the crossbar size shrinks by half, introducing extra area and power overheads. Considering the impact on latency, energy, and peripheral overheads, we select 256 as the crossbar size in our design.

V. Related Work

Many prior studies have proposed accelerator designs for genomic analysis targeting primary alignment algorithms, such as Smith-Waterman [2], [3], Needleman-Wunch [4], FM-index algorithm [5]–[8], and BLAST [23]. Among them, some are PIM designs based on non-volatile memories such as ReRAM [4], [5], ReRAM-based content-addressable memory (CAM) [2], [3], and SOT-MRAM [8], while others are FPGA-based designs [6], [23] or CMOS-based ASIC [7]. A prior work [9] has pointed out that the bottleneck among the entire genomic analysis pipeline is dedicated to Indel Realignment, so they propose an FPGA-based design to accelerate Indel Realignment rather than the primary alignment stage. Their design achieves $81.3\times$ speedup compared to a single SIMD CPU, outperforming the GPU-based design [24], [25] that shows only $3\times$ speedup over CPU when performing similar calculations. As far as we know, no other PIM-based design aims at accelerating the critical Indel Realignment algorithm. To our best knowledge, RePAIR is the first ReRAM-based PIM accelerator designed to achieve significant performance improvement and energy savings for Indel Realignment.

VI. Conclusion

In this paper, we propose a ReRAM-based PIM accelerator to accelerate Indel Realignment. We design several optimization schemes for scheduling and mapping to significantly increase the throughput while lowering the power consumption by reducing the hardware resource requirements and activating fewer WLs/BLs per array. To our best knowledge, this is the first work that utilizes a ReRAM-based PIM design to accelerate such a critical stage within the genomic analysis pipeline. Our design provides $7443\times$ speedup and consumes $27211\times$ less energy over the GATK3.8 benchmark running on an 8-thread CPU system, outperforming the state-of-the-art FPGA-based acceleration.

Acknowledgement

This work was supported in part by research grants from the Ministry of Science and Technology of Taiwan (MOST 109-2221-E-001-012-MY3).

References

[1] J. F. Hopkins *et al.*, “Mitochondrial mutations drive prostate cancer aggression,” *Nat. Commun.*, vol. 8, no. 1, p. 656, Sep 2017.
[2] R. Kaplan *et al.*, “Bioseal: In-memory biological sequence alignment accelerator for large-scale genomic data,” in *SYSTOR*, 2020, pp. 36–48.
[3] R. Kaplan *et al.*, “A resistive CAM processing-in-storage architecture for DNA sequence alignment,” *IEEE Micro*, vol. 37, no. 4, pp. 20–28, 2017.
[4] S. Gupta *et al.*, “RAPID: A ReRAM processing in-memory architecture for DNA sequence alignment,” in *ISLPED*, 2019, pp. 1–6.
[5] F. Zokaee *et al.*, “FindeR: Accelerating FM-index-based exact pattern matching in genomic sequences through ReRAM technology,” in *PACT*, 2019, pp. 284–295.
[6] J. Arram *et al.*, “Leveraging FPGAs for accelerating short read alignment,” *IEEE/ACM TCBB*, vol. 14, no. 3, pp. 668–677, 2016.
[7] Y. Wu *et al.*, “A 135mW fully integrated data processor for next-generation sequencing),” in *ISSCC*, 2017, pp. 252–253.
[8] S. Angizi *et al.*, “AlignS: A processing-in-memory accelerator for DNA short read alignment leveraging SOT-MRAM,” in *DAC*, 2019, pp. 1–6.
[9] L. Wu *et al.*, “FPGA accelerated INDEL realignment in the cloud,” in *HPCA*, 2019, pp. 277–290.
[10] A. Shafiee *et al.*, “ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars,” in *ISCA*, 2016, p. 14–26.
[11] P. Chi *et al.*, “PRIME: A novel processing-in-memory architecture for neural network computation in ReRAM-based main memory,” in *ISCA*, 2016, pp. 27–39.
[12] L. Song *et al.*, “Pipelayer: A pipelined ReRAM-based accelerator for deep learning,” in *HPCA*, 2017, pp. 541–552.
[13] B. Feinberg *et al.*, “Enabling scientific computing on memristive accelerators,” in *ISCA*, 2018, pp. 367–382.
[14] L. Song *et al.*, “GraphR: Accelerating graph processing using ReRAM,” in *HPCA*, 2018, pp. 531–543.
[15] A. Ankit *et al.*, “PUMA: A programmable ultra-efficient memristor-based accelerator for machine learning inference,” in *ASPLOS*, 2019, p. 715–731.
[16] M. A. DePristo *et al.*, “A framework for variation discovery and genotyping using next-generation DNA sequencing data,” *Nat. Genet.*, vol. 43, no. 5, p. 491, 2011.
[17] Y. Y. Chen *et al.*, “Balancing SET/RESET pulse for $> 10^{10}$ endurance in HfO$_2$/Hf 1T1R bipolar RRAM,” *IEEE TED*, vol. 59, no. 12, pp. 3243–3249, 2012.
[18] F. Nardi *et al.*, “Complementary switching in metal oxides: Toward diode-less crossbar RRAMs,” in *IEDM*, 2011, pp. 31–1.
[19] N. Muralimanohar *et al.*, “CACTI 6.0: A tool to model large caches,” *HP Laboratories*, 2008.
[20] N. Desai, “Design of high performance 16-bit Brent Kung adder using static CMOS logic style in 45nm CMOS NCSU free PDK,” *IRAJ IJASEAT*, vol. 1, no. 3, Jan. 2014.
[21] B. Murmann, “ADC performance survey 1997-2020,” [online], 2020.
[22] Intel Corp., “Intel Xeon gold 6144 processor 24.75M cache, 3.50 GHz.”
[23] L. Wienbrandt, “The FPGA-based high-performance computer RIVYERA for applications in bioinformatics,” in *CiE*, 2014, pp. 383–392.
[24] P. Klus *et al.*, “BarraCUDA - a fast short read sequence aligner using graphics processing units,” *BMC Res. Notes*, vol. 5, no. 1, p. 27, Jan 2012.
[25] Y. Liu and B. Schmidt, “CUSHAW2-GPU: Empowering faster gapped short-read alignment using GPU computing,” *IEEE Des. & Test.*, vol. 31, no. 1, pp. 31–39, 2014.