A 65nm RRAM Compute-in-Memory Macro for Genome Sequencing Alignment

Fan Zhang*, Wangxin He*, Injune Yeo*, Maximilian Liehr†, Nathaniel Cady†, Yu Cao*, Jae-sun Seo*, Deliang Fan*
*School of Electrical, Computer and Energy Engineering, Arizona State University, Tempe, AZ, USA
†College of Nanoscale Science and Engineering, SUNY Polytechnic Institute, Albany, NY, USA
E-mail: dfan@asu.edu

***Abstract*—In genomic analysis, the major computation bottleneck is the memory- and compute-intensive DNA short reads alignment due to memory-wall challenge. This work presents the first Resistive RAM (RRAM) based Compute-in-Memory (CIM) macro design for accelerating state-of-the-art BWT based genome sequencing alignment. Our design could support all the core instructions, i.e., XNOR based match, count, and addition, required by alignment algorithm. The proposed CIM macro implemented in integration of HfO_2 RRAM and 65nm CMOS demonstrates the best energy efficiency to date with 2.07 TOPS/W and 2.12G suffixes/J at 1.0V.**

***Index Terms*—RRAM, Compute-in-Memory, Genome Sequencing Alignment**

I. Introduction

Next-generation sequencing (NGS) technologies enable rapid and accurate determination of nucleotides (nt) sequence within genomes, empowering disease diagnostics, cancer risk assessment, tailored patient treatments, prenatal testing, and wide range of other personalized medicine approaches. NGS platforms can generate terabytes of DNA sequence (i.e., short reads) data in a single run. These short reads do not come with position information relevant to the overall genome and must be aligned to a reference genome before further genomic analysis or scientific discovery. However, the human reference genome is huge, containing approximately 3.2 billion nucleotide bases (A, C, G, T) [1]. Thus, a major challenge in sequencing is to map the short reads from NGS to the overall human reference genome.

State-of-the-art (SOTA) alignment processes still require hours or days to align large volumes of short read data, even using very powerful CPUs/GPUs [2]. This is mainly due to the off-chip bandwidth limitations and inefficiencies of moving big data between computation and memory units, i.e., the *memory-wall* challenge [3]. It is widely known that the bottleneck for the entire genomic analysis process is the memory- and compute-intensive DNA short reads alignment [4], [5]. To address the memory-wall challenge, Computing-in-Memory (CIM) has gained significant interest owing to its high energy efficiency and superior throughput [6], and has been widely investigated for accelerating AI applications [7], [8], but has not been applied considerably for genome alignment.

In this work, we propose and implement a resistive random access memory (RRAM) based CIM macro chip prototype for SOTA Burrows-Wheeler Transformation (BWT) [1], [2], [9], [10] based genome sequencing alignment. The designed CIM macro supports all core instructions, i.e., XNOR based match, count, and addition, required by alignment algorithms. As designed, this approach could work independently as a parallel 'alignment core' that could process local correlated reference genomic data to significantly improve system parallelism and throughput. Leveraging the multi-bit property of RRAM cells, our designed in-memory XNOR based match circuits are flexible to support both 1- and 2-bit per cell encoding of nucleotides (A, C, T, G). The CIM macro was implemented in a prototype chip that monolithically integrates HfO_2 RRAM and 65nm CMOS, achieving the best energy efficiency to date with 2.07 TOPS/W and 2.12G suffixes/J at 1.0V.

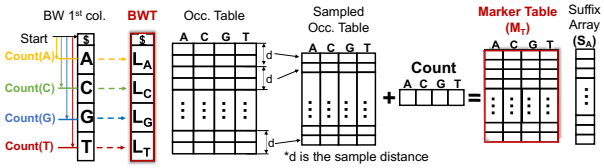


Fig. 1: Pre-computed tables for BWT based alignment.

II. Alignment-in-Memory Algorithm

In our prior works [4], [5], we have developed a CIM-friendly DNA short read alignment algorithm, called *alignment-in-memory* as shown in Algorithm 1, which recursively uses digital bit-wise logic functions to implement the fundamental computing core of BWT and FM-Index based genome alignment algorithm [1], [9]. Similar to the original algorithm, the one-time pre-computation is needed based on the reference genome S to construct required reference tables as shown in Fig. 1. The BWT is a reversible rearrangement of a character string. Exact alignment finds all occurrences of the short read R (m bp) in the reference genome S (n bp). Note that, only the BWT and Marker Table (M_T) are the primary genome alignment computations needed, and thus need to be stored in our CIM macro. Other tables, like $Occ.$ table and Suffix Array (S_A), are only related to pre- or post-processing of core alignment function. The BWT and M_T table mapping are one-time write and only memory-read based operations are needed during alignment computation, and are readily implemented with non-volatile RRAM technology.

The process described in Algorithm 1 is mainly implemented through the main $Bound(M_T, nt, id)$ procedure performed on BWT, which computes the updated interval bound (either low or $high$) value from M_T with bucket width d, input index-id and input nucleotide-nt. Such a procedure is iteratively used in every step of 'for' loop. To make the algorithm hardware-friendly for CIM platform, computations mainly leverage several logic functions, i.e., $XNOR\_Match$ and ADD. XNOR_Match conducts parallel *in-memory match* operation to determine if current input-nt matches with BWT elements stored in current memory, and then updates the $count\_match$ (i.e., counter) based on matching result (line-14

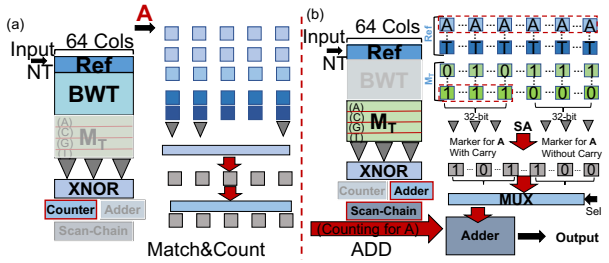


Fig. 2: Dataflow of alignment in one CIM macro.

to -18 in Algorithm 1). ADD performs 32-bit integer (determined by the 3.2 billion reference genome length) addition operation to implement '$marker + count\_match$', then the computed sum will be returned as the main $Bound$ function output (line-20). In summary, to implement all the alignment-related computations in Algorithm 1, the CIM platform needs to support parallel XNOR operations between input-nt and decoded BWT elements (line-15), count the XNOR results (line-16), read the marker from marker table (line-19), and add it to the current counter value (line-20).

Algorithm 1 Genome Alignment-in-Memory.

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

III. Computing-in-Memory RRAM Macro Design

A. Dataflow and mapping

Our preliminary work has developed *correlated data partition and memory mapping* methodology that could partition the BWT and M_T tables based on the target CIM macro memory size to guarantee each macro could work independently as an alignment-core to process within local memory array with correlated data partitions. Due to space limitations, we refer details of data partition algorithm to [4], [5]. Fig. 2 shows the data mapping and dataflow of alignment. For each CIM macro, the memory array is divided into three zones for storing and processing three data types: *i)* rows [0:3] defined as Ref where one whole row is programmed as the same genome nucleotide from [A,C,G,T] as a compute reference; *ii)* rows [4:15] storing the BWT partition for current CIM macro; *iii)* rows [16:63] storing the M_T table partition.

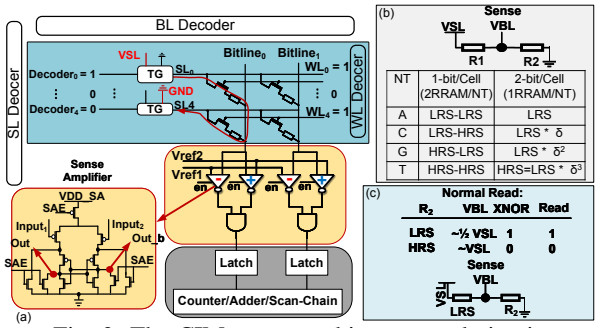


Fig. 3: The CIM macro architecture and circuits.

As illustrated in Fig. 2, the core alignment process in one macro requires two main stages: *match&count* and *ADD*. The match&count stage includes the parallel in-memory *XNOR_Match* and counting the matching result using a digital counter. For XNOR_Match operation, the first operand is the input-nt (A/ T/ C/ G), where the corresponding row in Ref region will be activated representing current $Bound$ function input. The second operand is a sub-list of BWT elements decoded by index-id and d (line-15 in algorithm). Therefore, in this stage, two decoded rows (one from Ref and one from BWT region) will be activated to implement parallel XNOR based match and count outputs (line-14 to -18). In the following ADD stage, the corresponding marker value from the M_T (line-19) will be fetched and added to the current counter (line-20) through a digital adder. Since the RRAM array only has 64 columns in the macro, the counting result will not be greater than 64 which can be represented in 6 bits. Performing a 32-bit addition with a 6-bit number in each local CIM macro is unnecessary, in our design, we leverage one 6-bit adder here and pre-calculate the bias based on the index of current CIM macro using similar BWT and M_T partition algorithm [4]. Note that, this pre-calculation is also one-time and saved within the M_T region for each type of nucleotide. Finally, the ADD result is returned as the main $Bound$ function output, which will be utilized during the processing of the next nucleotide in the same short read.

B. CIM Macro Architecture and Circuit Design

Fig. 3 shows the proposed architecture and circuits of one CIM macro to perform alignment operations. The computational array consists of one 64×64 RRAM array, SL/WL/BL decoder, sense amplifier (SA), counter, adder, transmission gate (TG), level shifter, etc. As described earlier, for XNOR_match operation, two rows will be simultaneously activated, as one example shown in Fig. 3(a). This forms a voltage divider circuit in each bitline (BL), where the BL voltage is determined by the two activated RRAM cells in the same column. Two complementary TGs controlled by SL decoder and drivers are used to provide the operating voltages. The first TG corresponding to the first XNOR operand, given input-nt in Ref region, connects to the VSL. While, the other one corresponding to the second XNOR operand, BWT, connects to the GND, for forming the voltage divider circuit as in Fig. 3(b). The voltage at the BL (VBL) should be around the middle of the supplied voltage (VSL) when the resistance of R1 is equal

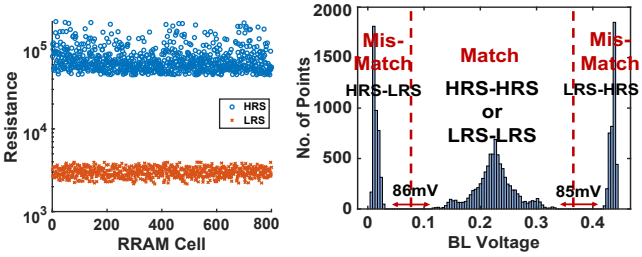
or very close to R2, meaning a 'match' is found. Otherwise, it is 'not matched'. To achieve such matching function, we design two sense amplifiers (SA) as voltage comparators per BL, where they share the same BL but with different reference voltages: Vref1 and Vref2. An AND logic gate is connected to the output of two SAs, so that it only outputs '1' when Vref1 $<$ VBL $<$ Vref2, thus implementing a XNOR based matching function. As described earlier, the example R1 and R2 here represent the two nucleotides being compared, as such, each column outputs '1' when the two nucleotides being compared are the same. With the operation being independent of other columns, it enables 64 parallel matching operations in one cycle. The proposed design supports both 1-bit/cell and 2-bit/cell XNOR_match, where the sense margin is mainly dependent on the RRAM's On/Off ratio, variation, resistance difference between different encoded levels and VSL. For the 1-bit/cell case, each nucleotide requires two adjacent RRAM cells for encoding, whereas in the 2-bit/cell case, each RRAM cell can be programmed into four different resistance levels: LRS, LRS$\times\delta$, LRS$\times\delta^2$, and HRS=LRS$\times\delta^3$ to represent the 4 different nucleotides.

In our design, to support independent RRAM programming, two complementary TGs are also present on BLs. During the XNOR_match operation, the BL is connected to SAs through the TGs. While, during RRAM cell programming, the BL is disconnected from SAs. The column decoder assigns the selected BL to an analog IO pad that provides Form/Set/Reset pulses to arbitrary RRAM cells in the array. Note that, VWL/VSL/VBL are directly connected to different analog IO pads to provide arbitrary pulses for RRAM device programming and testing.

The read of marker value stored in M_T memory region (line-19 in algorithm) leverages existing two rows activation scheme and XNOR_match circuit, but one of them must be in an all-LRS state. As illustrated in Fig. 3(c), it can be seen that when R1 is in the LRS state, the XNOR_match output is equivalent of reading R2's status. In both 1-bit and 2-bit per-cell cases, the first row is always programmed at the all-LRS state to encode nucleotide 'A', as shown in Fig. 3(b). Thus, the read operation is through activating the first row and the row needs to be read.

IV. Chip Measurement Result

Our prototype chip was fabricated using a custom 65nm CMOS process with integration of HfO_2 RRAM between M1 and M2 using a 300mm wafer platform at SUNY Polytechnic Institute. More detailed device-level RRAM characteristics and fabrication process were reported in the prior publication [11]. As shown in Fig. 8, we designed the automated process of the FORM/SET/RESET/READ operations for the RRAM array. Repeatable pulses are sent from SL/BL to BL/SL for each device during the programming process until the targeted resistance level is achieved, or the writing attempts reach the maximum limit. The WL is used for address indexing of the 1T1R cell in the RRAM array, and the typical value for programming amplitude (V), pulse width (PW), and gate control voltage (VWL) are listed below:



(a) Measured Cell Resistances (b) VBL Voltage

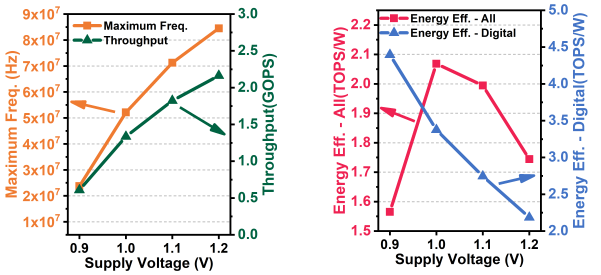
Fig. 4: RRAM resistance distribution and sensing voltage.

1) FORM: V_{form}=3.8V, PW=10μs, VWL=1.8V, repeat limit is 50 times; 2) SET: V_{set}=1.2V, PW=1μs, VWL=1.5V, target resistance value is $<3k\Omega$; 3) RESET: V_{reset}=3.3V, PW=100ns, VWL=3.3V, target resistance value is $>50k\Omega$; 4) READ: V_{read}=0.2V, VWL=3.3V.

In this work, we test and report the measurement results of the 1-bit/cell RRAM scheme, while the 2-bit/cell RRAM scheme remains as future work. Fig. 4(a) shows the measured RRAM LRS/HRS distribution across 5 test chips and the corresponding pattern match voltage distribution is shown in Fig. 4(b), where the center voltage distributions represent the BL voltage values with 'MATCH' results. For the XNOR_match operation, the VSL voltage is up-bounded to 0.45V. As we observed, a higher voltage than 0.45V may disturb RRAM resistance during inference operation.

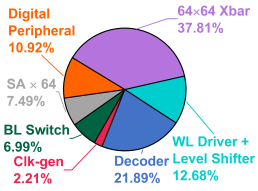
The chip's core power consumption comes from two main sources: analog input and digital power supply. The analog input feeds in from the SL through the given path of RRAM devices, with a fixed power supply at 0.45 V, to maximize the sensing margin while still preventing RRAM cells from destructive read operation. Analog power varies with test vectors from 150 μW to 400μW, as a result of different numbers of HRS and LRS in the circuit paths. In the energy efficiency calculation, we take 250 μW as the average analog power, where at this point the HRS and LRS cells are 50% each in the test vectors.

The digital power includes digital decoder, clock generator, digital driver for WL/SL/BL, and SAs. The digital power strongly correlates with the supply voltage and operating frequency. We performed a voltage sweep for the digital circuits from 0.9 V to 1.2 V, to explore the optimal voltage for the highest energy efficiency and the maximum frequency, and the results are shown in Fig. 5. In Fig. 5(a), we show the maximum frequency and throughput with voltage scaling. The maximum frequency (f_{MAX}) indicates the highest frequency at each supply voltage where all the circuit functions remain correct. The definition of throughput in this work is: $OPs/t \times f_{MAX}$, where OPs is the number of operations in one XNOR_match operation, which is 64 XNOR and counting (sum up 64 1-bit numbers), in total 128 for this work. t is the required number of cycles for the circuits to process the outputs from RRAM array, which is 5 in this work for SAs and the parallel adder. At 1.2V supply, we achieve the maximum frequency of 84.5 MHz and maximum throughput of 2.16 GOPS. As the supply



(a) Max. freq. & throughput (b) Energy Efficiency

Fig. 5: Frequency, throughput, and energy with voltage scaling.



(a) Macro area breakdown

Design	This Work
Technology Node	65nm
RRAM Type	1T1R HfO_2
RRAM Array Size	64 × 64
Core Design Area (mm^2)	0.1436
Operating Voltage (V)	0.9∼1.2
Operating Frequency (MHz)	23.7∼84.5
Energy Efficiency (TOPS/W)	2.07 (at 1.0V)

(b) Summary

Fig. 6: Area breakdown and chip summary.

voltage reduces, the frequency and throughput reduce largely linearly.

Fig. 5(b) shows the digital energy efficiency and overall (including digital and analog parts) energy efficiency. As the supply voltage reduces, the digital energy efficiency increases, while the analog energy efficiency degrades due to lower maximum frequency. The overall energy efficiency, which combines both digital and analog parts, reaches its maximum value of 2.07 TOPS/W at 1.0 V supply and a maximum frequency of 52.15 MHz.

Table I shows the comparison of our chip prototype design with four different types of genome alignment platforms: CPU/GPU as general purpose processors, FPGA [12] implementation, and ASIC design [10]. While CPUs/GPUs run at faster frequencies and have more on-chip memory, the 'memory-wall' limit their absolute throughput and energy efficiency. An FPGA-based implementation achieves higher performance due to its large-scale (8 FPGAs in this implementation) and dedicated dataflow graph. The only related prior CMOS ASIC design shows much improved performance, particularly in terms of throughput-to-area ratio, compared with CPUs/GPUs and FPGAs. Benefiting from the unique CIM architecture, our proposed CIM macro achieves the best

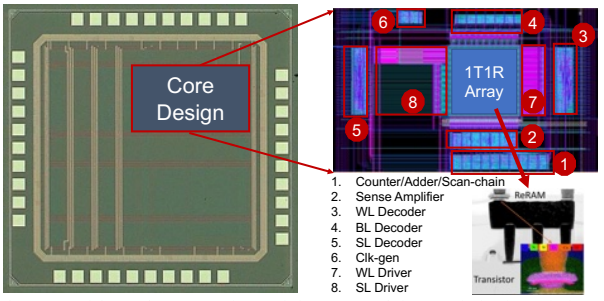


Fig. 7: Chip micrograph and layout with module breakdown.

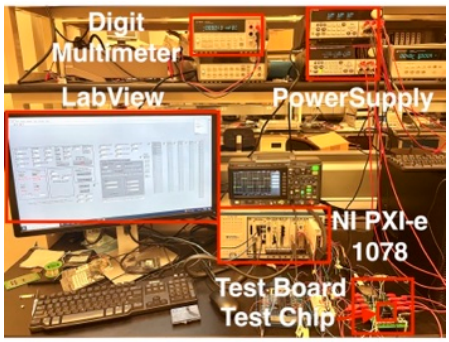


Fig. 8: Experimental test environment.

TABLE I: Comparison with prior related works.

Metrics	CPU [10] AMD Opteron 6128	GPU [10] NVIDIA Tesla M2075	FPGA [12]	ASIC [10] CMOS	Ours CMOS+RRAM
Technology	45nm	40nm	28nm	40nm	65nm
Die Size (mm^2)	14.3k	1.6k	14.8	7.84	0.1436
Power (W)	80	<200	247	0.135	0.01
Frequency (MHz)	2000	1150	200	200	84.5
On-Chip Memory (KB)	17,120	1,664	N/A	384	0.5(1-bit)/ 1(2-bit)
Throughput (suffixes/s)	6.9×10^4	8.3×10^5	1.5×10^8	5.1×10^6	2.12×10^8
Energy Efficiency (suffixes/J)	870	4200	6.2×10^5	3.7×10^8	2.12×10^9
Throughput-to-Area (suffixes/s/mm^2)	200	1600	420	6.4×10^5	1.47×10^9

performance in all aspects, particularly in energy efficiency and throughput-to-area ratio. Leveraging the high parallelism and reduced data movement of CIM architecture, our design achieves ∼41.6× higher throughput and ∼5.73× energy efficiency improvement against the SOTA CMOS ASIC design.

V. Conclusion

This work presents the first CMOS+RRAM CIM chip for accelerating genome sequencing alignment, showing orders of magnitude improvement in energy efficiency and throughput over CPUs/GPUs and prior non-CIM CMOS ASIC design.

Acknowledgment

This work is supported in part by the National Science Foundation under grants 1652866/2003749/2144751, and JUMP COCOSYS, an SRC program sponsored by DARPA.

References

[1] H. Li et al. Fast and accurate short read alignment with burrows–wheeler transform. *Bioinformatics*, 25:1754–1760, 2009.
[2] M. Alser et al. Accelerating genome analysis: A primer on an ongoing journey. *IEEE Micro*, 40(05):65–75, sep 2020.
[3] F. Wen et al. Openmem: Hardware/software cooperative management for mobile memory system. In *2021 58th DAC*, pp. 109–114.
[4] S. Angizi et al. AlignS: A processing-in-memory accelerator for DNA short read alignment leveraging SOT-MRAM. In *DAC*, pp. 1–6, 2019.
[5] F. Zhang et al. Aligner-d: Leveraging in-dram computing to accelerate dna short read alignment. *IEEE JETCAS*, 13(1):332–343, 2023.
[6] B. Li et al. Rram-based analog approximate computing. *IEEE TCAD*, 34(12):1905–1917, 2015.
[7] I. Yeo et al. Resistive memories stack up. *Nature Electronics*, 2022.
[8] A. Sridharan et al. A 1.23-GHz 16-kb programmable and generic processing-in-SRAM accelerator in 65nm. In *ESSCIRC*, 2022.
[9] M. Burrows et al. A block-sorting lossless data compression algorithm. *Digital Equipment Corporation technical reports*, 124, 1994.
[10] Y.-C. Wu et al. A 135-mW fully integrated data processor for next-generation sequencing. *IEEE TBioCAS*, 11(6):1216–1225, 2017.
[11] J. Hazra et al. Optimization of switching metrics for CMOS integrated HfO_2 based RRAM devices on 300 mm wafer platform. In *IMW*, 2021.
[12] J. Arram et al. Leveraging FPGAs for accelerating short read alignment. *IEEE/ACM TCBB*, 14(3):668–677, 2017.