A 65-nm RRAM Compute-in-Memory Macro for Genome Processing

Fan Zhang, *Student Member, IEEE*, Amitesh Sridharan, Wangxin He, Injune Yeo, *Member, IEEE*, Maximilian Liehr, Wei Zhang, Nathaniel Cady, *Member, IEEE*, Yu Cao, *Fellow, IEEE*, Jae-Sun Seo, *Senior Member, IEEE*, and Deliang Fan, *Member, IEEE*

***Abstract*— This work presents the first resistive random access memory (RRAM)-based compute-in-memory (CIM) macro design tailored for genome processing. We analyze and demonstrate two key types of genome processing applications using our developed CIM chip prototype: the state-of-the-art (SOTA) burrows–wheeler transform (BWT)-based DNA short- read alignment and alignment-free mRNA quantification. Our CIM macro is designed and optimized to support the major functions essential to these algorithms, e.g., parallel XNOR operations, count, addition, and parallel bit-wise and operations. The proposed CIM macro prototype is fabricated with monolithic integration of HfO_2 RRAM and 65-nm CMOS, achieving 2.07 TOPS/W (tera-operations per second per watt) and 2.12 G suffixes/J (suffixes per joule) at 1.0 V, which is the most energy-efficient solution to date for genome processing.**

***Index Terms*— Compute-in-memory (CIM), genome sequencing alignment, mRNA quantification, resistive random access memory (RRAM).**

I. Introduction

POWERED by high-throughput next-generation sequencing (NGS) technologies, the latest DNA sequencing method accurately decodes nucleotide (nt) sequences within genomes and assesses molecular activities in cells [1], [2]. This breakthrough supports various biomedical applications, including disease diagnostics [3], cancer risk assessment [4], personalized patient treatment [5], prenatal testing [6], and more. These biomedical applications heavily rely on fast and low-cost genome analysis [7], [8], [9].

Despite the significant reduction in sequencing costs due to the rapid growth of sequencing technologies, the current bottleneck lies in the compute-intensive nature of large-scale genome processing. For instance, DNA short-read (i.e., short DNA sequences) alignment involves aligning hundreds of millions of DNA short-reads to a 3.2 billion-length genome. Similarly, mRNA quantification entails measuring the expression level of hundreds of thousands of mRNA isoforms based on hundreds of millions of DNA short-reads for each patient sample. The existing genome analysis algorithms [10], [11], [12], [13], [14], [15], [16] still require several hours to days to process the large-scale genomic data, even on advanced computing architectures such as CPUs, GPUs, and ASICs, highlighting the persistent challenges posed by the data-intensive nature of these genome analysis processes.

Prior works have shown that genome processing is primarily bottlenecked by data movement and OFF-chip memory access [17], [18]. This memory wall bottleneck [19] is a well-known phenomenon plaguing traditional von-Neumann computers. A promising solution to the von-Neumann bottleneck is the compute-in-memory (CIM) paradigm where memory and compute are not treated as isolated elements. Instead, compute is interleaved inside memory for higher throughput and reduced data traffic [20]. Several prior works have reported significant improvement in genome analysis throughput with CIM architectures [17], [21], [22], [23], [24], [25]. However, most CIMs for genome processing have not been implemented on a custom chip.

The key contributions of this work are summarized below.

1) *First RRAM-based CIM chip prototype for genome short-read alignment and quantification.* To the best of the authors' knowledge, we demonstrate the first resistive random access memory (RRAM)-based CIM chip prototype designed and optimized for genome processing. We have demonstrated and evaluated two critical genome processing tasks: burrows–wheeler transform (BWT)-based DNA short-read alignment and mRNA quantification where each short-read ranges from 50 to

Manuscript received 7 December 2023; revised 12 April 2024; accepted 22 April 2024. Date of publication 14 May 2024; date of current version 28 June 2024. This article was approved by Associate Editor Danilo Manstretta. This work was supported in part by the National Science Foundation under Grant 2314591, Grant 2414603, Grant 2349802, and Grant 2342726; and in part by the Center for the Co-Design of Cognitive Systems (CoCoSys) under Joint University Microelectronics Program (JUMP) 2.0, a Semiconductor Research Corporation (SRC) program sponsored by Defense Advanced Research Projects Agency (DARPA). *(Corresponding author: Fan Zhang.)*

Fan Zhang, Amitesh Sridharan, and Deliang Fan are with the Department of Electrical and Computer Engineering, Johns Hopkins University, Baltimore, MD 21218 USA (e-mail: fzhang62@jh.edu; dfan10@jh.edu).

Wangxin He is with the School of Electrical, Computer and Energy Engineering, Arizona State University, Tempe, AZ 85287 USA.

Injune Yeo is with the Department of Electrical Engineering, Chosun University, Gwangju 61452, South Korea.

Maximilian Liehr and Nathaniel Cady are with the College of Nanotechnology, Science, and Engineering, University at Albany, Albany, NY 12222 USA.

Wei Zhang is with the Department of Computer Science, University of Central Florida, Orlando, FL 32816 USA.

Yu Cao is with the Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis, MN 55455 USA.

Jae-Sun Seo is with the School of Electrical and Computer Engineering, Cornell Tech, New York, NY 10044 USA.

Color versions of one or more figures in this article are available at https://doi.org/10.1109/JSSC.2024.3396429.

Digital Object Identifier 10.1109/JSSC.2024.3396429

300 nucleotides (nts). Our designed CIM macro supports all the core instructions required by the alignment and quantification algorithms, i.e., XNOR-based match, parallel AND operations, count, and addition. As designed, each CIM macro works independently as a parallel "alignment/quantification core" that could process locally correlated genomic data to significantly improve system parallelism and throughput. The CIM macro is implemented in a prototype chip that monolithically integrates HfO_2 RRAM and 65-nm CMOS, achieving the best energy efficiency to date with a throughput of 2.07 TOPS/W (tera-operations per second per watt) and 2.12 G suffixes/J (suffixes per joule) at an operating voltage of 1.0 V.

2) *Performance benchmarking:* We conduct a comprehensive evaluation and benchmarking of the performance and energy efficiency of our design with a spectrum of recent advancements in the field. For the short-read alignment, our benchmarking includes a range of technologies: traditional CPU and GPU systems, CMOS-based ASIC designs [26], [27], and FPGA [28]. Our design demonstrates a remarkable performance improvement, achieving up to a 41-fold increase in throughput compared with these conventional systems. Moreover, in terms of energy efficiency, our design outperforms the existing solutions by up to $\sim$5.73×. For mRNA quantification, we conduct the analysis by adapting and mapping our quantification-in-memory algorithm to both our CIM design and other state-of-the-art (SOTA) non-volatile memory based CIM designs that are tailored for bit-wise logic operations. Our CIM design not only demonstrates higher energy efficiency but also achieves the best normalized throughput against column parallelism.

II. Background and Motivation

A. Short-Read Alignment

The sequencing data obtained from a single patient sample include hundreds of millions of DNA short-reads, each ranging from 50 to 300 nts (A, T, C, G) in length. These short-reads lack positional information. To address this, aligning these short-reads with the reference genome becomes crucial for most genomic analyses [29], [30]. However, this alignment task is challenging due to the extensive nature of the reference genome. Despite the development of numerous sequence alignment algorithms over the past decades, even the most efficient ones, like BWA [15], [31] or Bowtie2 [12], still require hours or days for alignment. The extended processing time for aligning such large amounts of data poses a challenge, limiting the widespread use of genomic information in disease diagnosis and prognosis within clinical and hospital settings.

B. mRNA Quantification

mRNA transcription is a fundamental aspect of cellular function. The level of mRNA transcription directly influences the amount of protein synthesized by a cell, and it plays a central role in various biological processes, offering insights into health, disease, and therapeutic interventions [32], [33]. Therefore, the efficient and accurate quantification of mRNA transcription levels becomes crucially important.

The current alignment-free technique focuses on determining the mRNAs from which the short-reads are generated rather than their exact locations [34]. Despite the efficacy of this strategy in preserving accuracy, there remains a need to map each short-read to hundreds of thousands of mRNAs for quantification, demanding substantial computational resources. To tackle this challenge, it is essential to develop an efficient and fast hardware accelerator, specifically designed for the compute- and data-intensive demands of the alignment-free mRNA quantification process.

C. Acceleration of Alignment

There have been several prior works that accelerate short-read alignment in CPUs [35], GPUs [36], [37], [38], and ASICs [18], [26], [27], [39], [40]. Notably, there have also been simulation-based efforts in the realm of CIMs [17], [22], [23], [25].

1) Alignment in CPUs and GPUs: The SOTA CPU-based implementation [35] provides end-to-end acceleration through instruction-level re-organization that promotes more efficient caching. It also uses SIMD instructions to amortize control overhead and mask the memory wall through aggressive prefetching. These techniques provide a speedup of up to 2.4× −3.5× over the existing CPU-based software implementations. Liu and Schmidt [36] accelerate alignment on Nvidia GPUs using CUDA. It offers a heterogeneous CPU–GPU acceleration and claims a speedup of 1.6× −2.7× over the CPU-only implementations.

2) Alignment in ASICs: A simulation-based ASIC implementation [41] has shown $\sim$24× improvement compared with software baselines for aligning long sequence reads. Another simulation-based ASIC, NvWA [18] addresses the diversity and scheduling problem between seeding and seed-extension phase. It provides speedups of over 493× compared with CPU baseline, 200× over [37] and 12× over [40]. As for real silicon implementations, [26] represents one of the first works in a 40-nm prototype chip. It achieves >16× throughput improvement over GPU. Another work [27] based on 28-nm CMOS implements an SoC that performs read mapping along with other steps in the genome sequencing pipeline.

3) Alignment in CIMs: CIM has been established as a promising solution to accelerate data-intensive tasks by reducing dominating data traffic. Li et al. [17] accelerate alignment in DRAMs and offer >1820× speedup over conventional methods. Whereas [22], [23], [25] accelerate alignment in non-volatile memories (NVMs), reaping the benefits of both CIM speedup and non-volatile storage. Prior works like [22], [23] accelerate alignment by forming every RRAM device several times for each short-read alignment and using multi-bit analog-to-digital converters (ADCs) which have high error rates. This is not tolerable, especially for alignment where error resilience must be less than 0.1% (software baseline for short-read alignment). To alleviate this bottleneck, we design our macro in a mostly digital fashion without

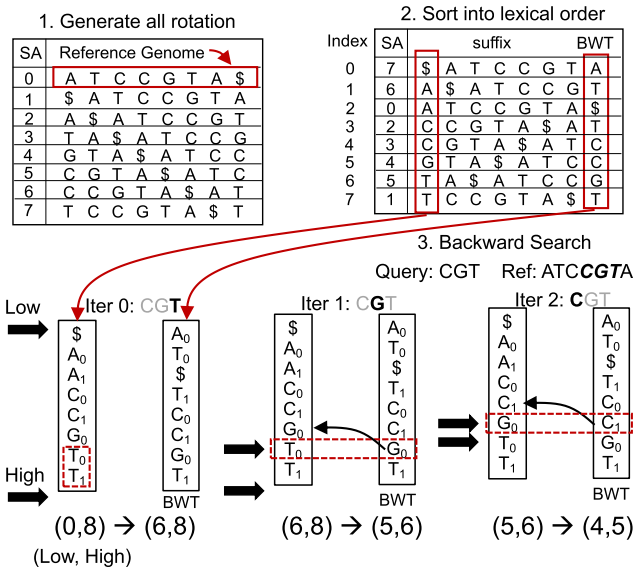


Fig. 1. BWT-based genome alignment steps. Given the reference genome "ATCCGTA" finds the occurrence of query "CGT" with backward search.

the need for forming during alignment. All prior NVM-CIM-based alignment works [22], [23], [25] are simulation-based implementations that do not represent real-world metrics. To our knowledge, we are the first to demonstrate working RRAM/CMOS-based CIM for short-read alignment on real silicon implementation.

III. Genome CIM Algorithms

A. Alignment-in-Memory Algorithm

Fig. 1 shows the original BWT-based backward search method to find the occurrence in a reference sequencing. It consists of three main steps.

1) First, given a reference genome sequence, it begins by generating all possible rotations of this sequence through cyclic rightward shifts, assigning each rotation an increasing index in the suffix array (SA) column.
2) Next, it lexicographically sorts these rotations and assigns new indices. The last column of this sorted array is known as the BWT [15].
3) Finally, using the first and last columns, the occurrence of a substring can be located through backward searching.

Fig. 1 demonstrates an example of the process of how to search for the occurrence of "CGT" from the reference sequence "ATCCGTA$," where "$" denotes the end of the sequence. Two pointers, "low" and "high," are initialized to the beginning (0) and end (8) of the first column, respectively. After the backward search, the occurrence location is indicated by these two points.

In our prior works [25], [30], we have developed a CIM-friendly DNA short-read alignment algorithm, called *alignment-in-memory* as shown in Algorithm 1. Compared with the original BWT algorithm [15], [31], we have replaced the operators with equivalent CIM-friendly bit-wise logic operations. Similar to the original algorithm, the pre-computation is needed based on the reference genome S to construct required

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:
$low \leftarrow 0$, $high \leftarrow |S| - 1$
Step-2. Backward Search:
for $i := |R| - 1$ to 0 **do**
  $low \leftarrow$ ***Bound***$(M_T[\lfloor low/d \rfloor], R[i], low)$
  $high \leftarrow$ ***Bound***$(M_T[\lfloor high/d \rfloor], R[i], high)$
  if $low \geq high$ **then**
    break & return 0 ▷ there is no exact alignment
  end if
end for
Step-3. Get matched positions from the stored suffix array based on a search result:
for $j := low$ to $high - 1$ **do**
  $positions \leftarrow$ **MEM**$(S_A[j])$ ▷ Read positions from Suffix Array memory
end for

Define procedure Bound:
Procedure: ***Bound***(M_T, nt, id) ▷ compute matched interval
$C_M \leftarrow 0$ ▷ Initialize variable C_M(Count Match)
for $j := 0$ to $j < (id \bmod d)$ **do** ▷ count number of nt within the BWT region
  if ***XNOR_Match***$(nt, BWT[id - (id \bmod d) + j]) == 1$ **then**
    $C_M = C_M + 1$
  end if
end for
$marker \leftarrow$ ***MEM***$(M_T[\lfloor id/d \rfloor], nt])$ ▷ Read Marker Table value
return ***ADD***$(marker, C_M)$
end Procedure

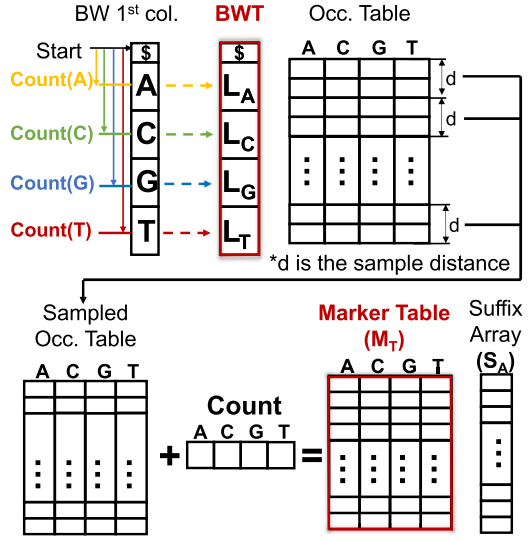


Fig. 2. Pre-computed tables for BWT-based alignment.

reference tables as shown in Fig. 2. This pre-computation is a one-time effort as different query sub-strings use the same BWT during the backward search. The BWT is saved as occurrence table (denoted as Occ.). To reduce storage needs, Occ. is sampled for every "d" rows ($d = 32$ in our macro design). Therefore, to accurately determine the exact occurrence, it is necessary to count the nts within each "d" interval. The BWT and M_T table mapping require a one-time write, and only memory-read-based operations are needed during alignment computation. Therefore, they are well-suited for CIM leveraging NVMs, such as RRAM.

The process described in Algorithm 1 is mainly implemented through the Bound(M_T, nt, id) procedure, which iteratively updates interval bound pointers (either low or high). To make the algorithm hardware-friendly for the CIM platform, all the computations leverage certain bit-wise logic functions, i.e., **XNOR_*Match*** and ***ADD***. **XNOR_*Match*** conducts parallel *in-memory match* operation to determine

Algorithm 2 mRNA Quantification-in-Memory

Require: : Construct index_table-T for each genome S. The index table consists of K-mers and associated k_comp classes.
input: Genome Short-Read R
output: Compatible transcripts of short-read R in reference genome S
1: Initialize $result = \textbf{\textit{ones}}(t, m)$ ▷ t is the number of index tables, m is the number of transcript.
2: $input = R[i = 0{:}j = k]$ ▷ k is the length of K-mer.
3: **for** $input$ **in** $short\_read$ **do**
4: Initialize $k\_mer\_idx = \textbf{\textit{ones}}(t, n)$ ▷ n is the number of K-mers in each index_table.
5: **parfor** t_i **in** $length(T)$ **do** ▷ Partition the data and Ops for high parallelism.
6: **for** i **in** $[0 : k]$ **do**
7: $xnor\_result[t\_i] = \textbf{\textit{XNOR\_Match}}(input[i], T[t\_i][k\_mer][i])$
8: $k\_mer\_idx[t\_i] = \textbf{\textit{AND}}(k\_mer\_idx[t\_i], xnor\_result[t\_i])$
9: **end for**
10: $k\_comp = \textbf{\textit{MEM}}(T[t\_i][k\_comp][K\_mer\_idx])$ ▷ Read the K-comp.
11: $result[t\_i, :] = \textbf{\textit{AND}}[result(t\_i, :], k\_comp)$
12: **end parfor**
13: $input = short\_read[i + 1 : j + 1]$
14: **end for**
15: **Return** $result$ ▷ result indicates the compatible transcripts of all genes.

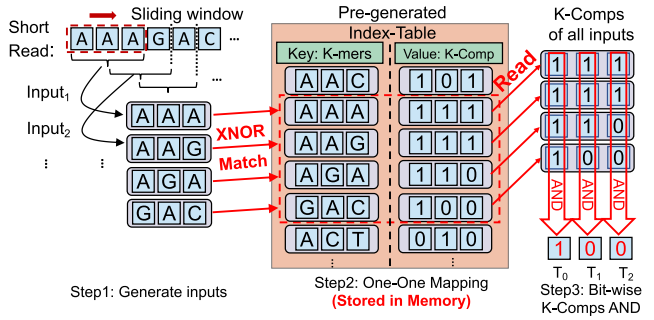


Fig. 3. Example steps in mRNA quantification-in-memory.

whether current input nt matches with BWT elements stored in the current memory array, and then updates the C_M (Count_Match) based on matching result (lines 14–18 in Algorithm 1). ***ADD*** performs 32-bit integer addition to implement "marker + C_M," then the computed sum will be returned as the main ***Bound*** function output (line 20). For a more comprehensive understanding of the textitalignment-in-memory algorithm and its adaptation of the BWT backward search, we encourage readers to refer to our prior publication [25], [30].

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).

B. Quantification-in-Memory Algorithm

The CIM-friendly mRNA quantification-in-memory algorithm is depicted in Algorithm 2 and Fig. 3, motivated by our prior work [42]. It involves three key steps.

1) Index table construction.
2) Input generation and K-mer **XNOR_*Match***.
3) K-comp **AND** to identify the transcript.

The human genome comprises thousands of genes, and each gene is represented by one index table. Every index table contains two parts: *K-mers* (a substring of length k from a longer sequence) and *K-comp* (set of transcripts that share a common k-mer) (requirements in Algorithm 2). All the transcripts' sequences of a gene are sliced into substrings of length "k," termed "k-mers," starting from each position; and for each K-mer, the K-compatibility ("K-comp") classes are defined based on its occurrence in the transcript. The K-comps is represented as a binary 1-D vector: "1" denotes the presence of the k-mer, while "0" indicates its absence. It is important to note that the construction of this index table is a one-time process for each gene. Once constructed, these tables are stored within the CIM platform for efficient access and utilization.

After the construction, a sliding window, matching the length of the K-mers, generates inputs from every short-read (lines 2 and 13). Each input is then processed against every gene's index table to identify exact matches using a bit-wise XNOR logic implemented in hardware (lines 5–8).

When a match is found in an index table, its corresponding K-comp value is recorded and processed (lines 10 and 11). If no match is found, the short-read does not belong to any transcript in this gene and is discarded for this index table by returning an all-zero vector. This all-zero vector is generated by the MEM function when the requested k_mer_idx is an all-zero vector, indicating that the requested K-comps do not exist.

To illustrate the process, one example is presented in Fig. 3. K-mer length is 3 in this example. Assuming this gene has three transcripts, resulting in the K-comp length as 3. Each input will be fed into the pre-generated index table to find the exact match and its corresponding K-comp. Bit-wise **AND** is then performed on all the selected K-comps to produce the final output. In this example, based on the final AND output of "100," transcript 0 is found as the compatible transcript. Note that, the final output may have multiple "1"s, indicating more than one transcript is compatible.

The primary operations in our quantification-in-memory algorithm are K-mer matching (via **XNOR_*Match***) and **AND** for matched K-comps. The matching operations across different index tables are independent, and each CIM array can act as a separate process engine, fully exploiting the parallelism of CIM architecture. For a more detailed explanation and analysis of the CIM-friendly mRNA quantification-in-memory algorithm, refer to [34] and [42].

In summary, to facilitate alignment and quantification, the CIM design must support the following major operations: **XNOR_*Match***; regular memory access and read (***MEM***); ***ADD***; and bit-wise **AND**. It is worth noting that unlike many RRAM crossbar array-based machine learning (ML) accelerator designs that leverage high parallelism to efficiently implement vector–matrix multiplication or dot-product operations, genome alignment and quantification require very different logic operations and workflows that cannot be simply modeled as dot products. Therefore, a new design specifically targeting genome processing is desired to fully leverage the benefits of CIM.

IV. CIM RRAM Macro Design

Fig. 4 shows the proposed architecture and circuits of one CIM RRAM macro for alignment and quantification operations. The computational array consists of one

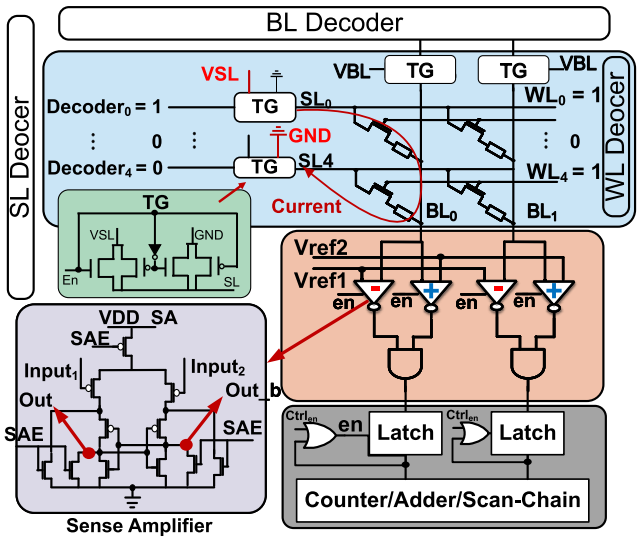


Fig. 4. CIM macro architecture and circuits.

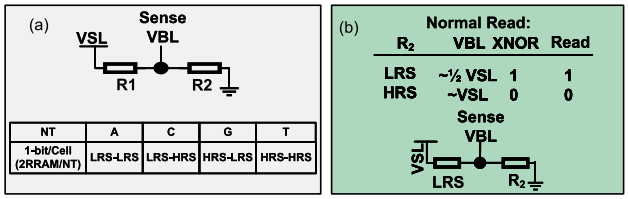


Fig. 5. (a) Circuits for XNOR operation. (b) Read operation (MEM) through existing circuits.

64 × 64 RRAM array, SL/WL/bitline (BL) decoder, sense amplifier (SA), counter, adder, transmission gate (TG), level shifter, etc. Note that in this work, we mainly treat each RRAM device as single-level cell (SLC) with two distinct resistance states, i.e., high-resistance state (HRS) and low-resistance state (LRS).

A. Parallel XNOR Operation (XNOR_Match)

For **XNOR_*Match*** operation, two rows storing the two operands will be simultaneously activated, as one example shown in Fig. 4. This forms a voltage divider circuit in each BL, where the BL voltage is determined by the two connected RRAM cells in the same column, as shown in Fig. 5(a). Two complementary TGs controlled by source-line (SL) decoder and drivers are used to provide the operating voltages. When the decoder's output is set to "1," the TG at the corresponding row connects the selected SL to VSL, and when the output is set to "0," the corresponding SL is connected to GND.

During XNOR operation, two rows are activated simultaneously where the first row corresponds to the first XNOR operand, given input nt in Ref region through connecting the corresponding TG to the VSL. The second activated row corresponds to the second XNOR operand, i.e., BWT for alignment application or K-mers for quantification application. For the second activated row, the TG is connected to GND, for forming the voltage divider circuit [Fig. 5(a)]. When the resistance of R1 is equal or very close to R2, the voltage at the BL (VBL) should be around half of VSL, meaning a "match" is found. Otherwise, the deviated VBL represents "not matched."

To detect the VBL around half of VSL for implementing such a matching function, we design two SAs 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 an XNOR-based matching function. As described earlier, the examples R1 and R2 here represent the two operands being compared, where each column outputs "1" when the two operands being compared are identical. With the operation being independent of other columns, it enables 64 parallel XNOR operations in one cycle. For the in-memory **XNOR_*Match*** operation discussed above, the sensing margin is mainly dependent on the RRAM's ON/OFF ratio, variation, and VSL. Given our SLC RRAM devices, each nt requires two RRAM cells to represent the four different types of nts, e.g., "A" (LRS-LRS), "C" (LRS-HRS), "G" (HRS-LRS), and "T" (HRS-HRS), as shown in Fig. 5(a). Therefore, two adjacent RRAM cells on the same row are used to represent a single nt. The 64 parallel XNOR operations are transferred to 32 nts (interval "d" described in Section III-A) matching operation with the peripheral circuits in counter.

In our design, to support independent RRAM programming, two complementary TGs are also present on BLs. During the **XNOR_*Match*** operation, BL is connected to SAs through the TGs. 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. VWL, VSL, and VBL are directly connected to different analog IO pads to provide arbitrary pulses for RRAM device programming and testing.

B. Normal Memory Access (*MEM*)

As required by both the alignment and quantification algorithms discussed earlier, the proposed design also needs to read data from memory for further processing. Instead of having a separate read circuit, we share the two SAs used for the CIM mode, also for such ***MEM*** function. We use the existing CIM scheme of activating two rows simultaneously and the existing **XNOR_*Match*** circuit. However, it is crucial that all RRAM cells in one of these activated rows are programmed in LRS. As illustrated in Fig. 5(b), it can be seen that when R1 is in the LRS state, the **XNOR_*Match*** output is equivalent to reading R2's status. As explained in Section V, the first four rows of the RRAM macro are always reserved as references representing four different types of nt, corresponding to current input nt. In this setup, the first row is always programmed at the all-LRS state to encode nt "A." Thus, the read operation could be implemented through activating the first row and the row needs to be read using the existing peripheral circuits.

C. Other Peripherals for *Count/ADD/*AND

The other required "Counter" and "Adder" for the alignment task are implemented with digital circuits, subsequent to the

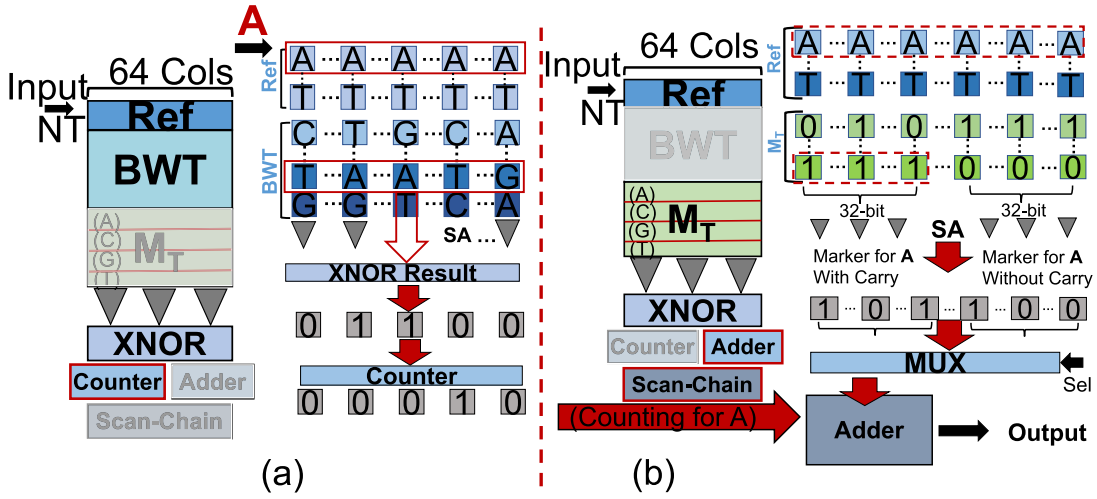


Fig. 6. Dataflow of alignment in CIM macro. (a) Match&count. (b) ADD.

dual SAs. A latch is associated with each BL and is positioned after the voltage comparator to temporarily store the result.

For the quantification task, an extra "and" operation is needed. To minimize the add-on logics and provide the switch knob to disable such function in alignment workload, we propose to repurpose the latches for the dual SAs to execute AND operation. It could be achieved by connecting the output of each latch to its enable port through an OR gate. This configuration not only allows the latch to store the current **XNOR_*Match*** result but also enables a bit-serial AND function, continuously performing the AND operation between the incoming input and the previously stored result. More importantly, the or gate between the latch's output and enable port provides the flexibility to activate or deactivate this and function as needed. For the alignment task, the and function is not required. In this scenario, the decoder emits a control signal (ctrl_en) to the or gate, ensuring that the latch's enable port is always active, thereby functioning as a standard latch. Conversely, during the quantification task, the and function is activated after each **XNOR_*Match*** and ***MEM*** operation. Here, the decoder generates the ctrl_en signal only for the initial cycle to initialize the latch. From the second cycle onward, the latch's behavior is controlled by its previous output. If any input to the latch is "0," the latch becomes disabled, maintaining a "0" state regardless of subsequent inputs. This mechanism mirrors the principle of an and operation, where the result is "0" if any input is "0."

With the aforementioned innovations, the proposed CIM RRAM macro can execute all the five essential functions required for alignment and quantification: **XNOR_*Match***, ***MEM***, ***Count***, ***ADD***, and **AND**.

V. Dataflow and Mapping

A. Alignment-in-Memory Dataflow and Mapping

Our preliminary work [25] has developed *correlated data partition and memory mapping* methodologies that could partition the BWT and M_T tables based on the target CIM macro memory size and map them onto RRAM arrays to eliminate frequent data movement. Such correlated data partition also guarantees that each macro could work independently as an alignment core to process within the local memory array with correlated data partitions. We refer details of the data partition algorithm to [25] and [30]. When querying a short-read against the entire reference genome, the CIM-friendly alignment capitalizes on the independence of these CIM macros and their inherent high parallelism. This setup enables simultaneous search of short-reads across the entire reference genome, with each CIM macro efficiently processing its allocated segment within the local memory array. As a result, the overall search operation is fast and efficient, leveraging distributed data partitions across macros.

Fig. 6 shows the data mapping and dataflow of alignment operation in one CIM macro. For each CIM macro, the memory array is divided into three zones for storing and processing three data types: 1) rows [0:3] defined as Ref where one whole row is programmed as the same genome nt from [A,C,G,T] as a compute reference; 2) rows [4:15] storing the BWT partition for the current CIM macro; and 3) rows [16:63] storing the M_T table partition for current CIM macro. With the 1-bit per cell encoding scheme, a 64 $\times$ 64 crossbar array can store the BWT and M_T for a fragment of the reference genome up to 384 nts long. Reference genomes longer than this will be partitioned and mapped onto multiple RRAM macros for parallel processing.

As illustrated in Fig. 6, 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***, 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 1). Therefore, in this stage, two decoded rows (one from Ref and one from the BWT region) will be activated to implement parallel XNOR-based match and count outputs (lines 14–18). An example is given in Fig. 6(a), for the task of finding the match between the input nt "A" and a BWT row "TAATG." The first row in

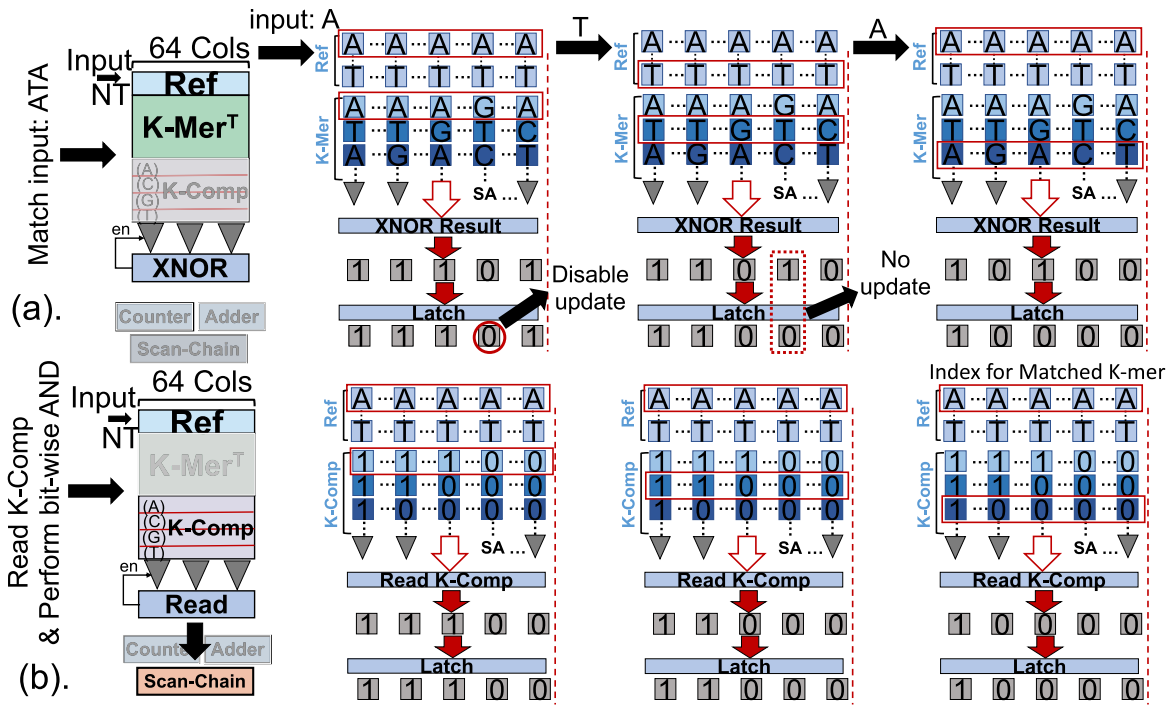


Fig. 7. Mapping and dataflow of quantification in CIM macro. (a) **XNOR_*Match*** between input nts and local stored K-Mers. (b) Serial bitwise AND operation among **XNOR_*Match*** results to identify the compatibility.

the reference part of the macro (stores all "A") and the corresponding BWT row are activated. This activation scheme resulting XNOR operation produces "01 100." The XNOR result is sent to the digital counter, which converts this pattern into its binary-encoded representation of "00 010." In the following ADD stage [Fig. 6(b)], the corresponding marker value from M_T (line 19) will be fetched and added to the current counter result (line 20) through a digital adder. Finally, the ***ADD*** result is returned as the main ***Bound*** function output, which will be used during the processing of the next nt in the same short-read.

B. Quantification-in-Memory Dataflow and Mapping

Similar to the alignment process, the CIM-friendly mRNA quantification also capitalizes on the independent nature and high parallelism of CIM macros (as outlined in line 5 of Algorithm 2). Each gene constructs its own unique index table. During the processing of a short-read, this read is concurrently distributed to all the CIM macros. Each macro, holding a distinct index table, executes the following steps to process with its unique gene within the local memory array. The ON-chip RRAM array is strategically partitioned into two distinct sections, one stores the pre-computed index table's K-mers and the other holds the corresponding K-comp. K-mers are stored in a transposed orientation along column-wise marked as K-Mer^T in Fig. 7. Each K-mer occupies the same column, where each row in the section stores the same indexed nt among different K-mers. It enables us to perform the bit-wise **XNOR_*Match*** among K-mers in parallel. K-comps are stored along word-lines in a binary format, where each column corresponds to a transcript. In this format, the binary value in each column indicates whether the associated K-mer is part of the corresponding transcript. For example, a "1" in the zero-indexed column signifies that the K-mer of this K-comp is present in transcript T_0.

When the short-read is received, the first step is to generate inputs with a sliding window to guarantee the input has the same length as the K-mer for the following **XNOR_*Match***. With the generated input, we activate two rows in the sub-array to perform the **XNOR_*Match***. Since the K-mers are stored in a transposed format and the XNOR operation is executed bit-wise, it requires K cycles to identify a matched K-mer, where K is the length of the K-mer.

As illustrated in Fig. 7(a), the input "ATA" takes three cycles to locate the matching K-mer in the sub-array. In each cycle, we simultaneously activate two rows: one row from the reference section, representing the input nt, while the other row holds the nt of K-mers at various indices. To optimize parallel computing efficiency, all the BLs are simultaneously activated to perform **XNOR_*Match*** between the nt from the current input and the selected K-mers in parallel, processing each bit in tandem. As previously discussed, during the initial cycle, the outcomes of these **XNOR_*Match*** are captured in latches, activated via the Ctrl_en signal. During the subsequent cycles, these latched results are fed back into the system, controlling the enable pins of the latches. This setup ensures that when a latched result switches to "0," the corresponding latch becomes disabled and ceases to update. This feedback mechanism, operating through the enable signal, essentially acts as a serial bitwise **AND** operation among the **XNOR_*Match*** results (line 8). After three cycles, the final result of "10 000" is obtained, which indicates that one match is found, and "ATA" is stored in the first column. Thus, the K-comp stored in the first row of the corresponding K-comp sub-array is selected

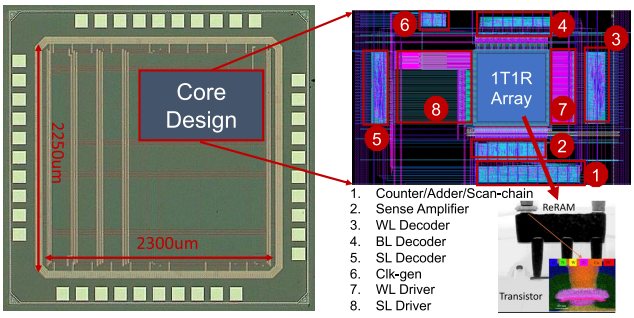


Fig. 8. Chip micrograph and layout with module-level breakdown.

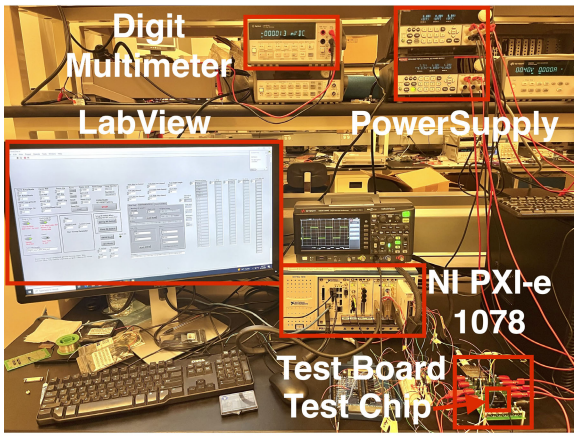


Fig. 9. RRAM chip testing environment.

for the subsequent **AND** operation. The corresponding *K*-comp value "11 100" will be read out and saved into the latches. The decoder sequentially activates the rows in the *K*-comp section based on the previous **XNOR_*Match*** result. The same **AND** operations are conducted by the latches, using the previous **AND** result stored within. When all the inputs are processed, the final result is saved in latches, indicating the transcript compatibility of the current short-read. Fig. 7 provides an example of processing three inputs. However, it is possible that an exact match might not exist within the *K*-mer array. In such scenarios, the result of the **XNOR_*Match*** process would yield an all-zero output. This outcome is detected by the system. Consequently, no *K*-comp is activated for subsequent **AND** operations, and the decoder directly issues a "Not Found" signal in response to this non-match situation.

VI. Results and Analysis

A. Chip Measurement Result

Our prototype chip (Fig. 8) was fabricated using a custom 65-nm CMOS process with monolithic integration of HfO_2 RRAM between metal 1 (M1) and metal 2 (M2) using a 300-mm wafer platform at the Albany NanoTech Complex in Albany, NY. Briefly, a standard 65-nm CMOS process flow was used for transistor fabrication, with RRAM bottom electrode, switching layer, oxygen exchange layer, and top electrode implemented at the interface of M1 and via 1 (V1). RRAM integration was performed in the same 300-mm processing line as the CMOS levels, and a standard 65-nm back end of line (BEOL) processing was performed after implementing the RRAM layers. More detailed device-level RRAM characteristics and fabrication processes were reported in the prior publication [43].

As shown in Fig. 9, 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 one transistor and one RRAM (1T1R) cell in the RRAM array, and the typical values for programming voltage amplitude (V), pulsewidth (PW), and gate control voltage (VWL) are listed in Table. I and summarized below.

1) FORM: $V_{\text{form}} = 3.8$ V applied from SL to BL, PW = 10 μs, VWL = 1.8 V, and repeat limit is 50 times; 2) SET:

TABLE I
RRAM Cell Programming Parameters

	VSL	VBL	VWL	Pulse Width	Max. Repeat
Form	3.8V	GND	1.8V	10μs	50 Times
Set	1.2V	GND	1.5V	1μs	
Reset	GND	3.3V	3.3V	100ns	

* VSL/VBL/VWL are voltages applied on SL(Source-line), BL(Bit-line), and WL(World-line).

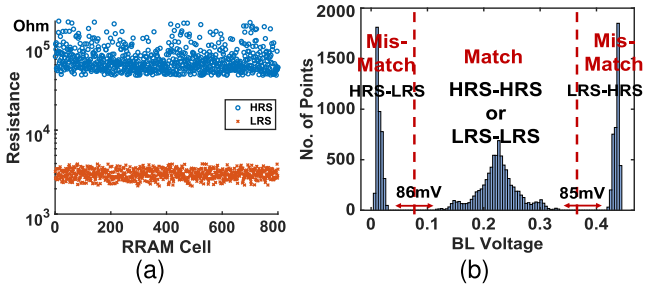


Fig. 10. (a) Distribution of measured RRAM resistances. (b) BL sensing voltages for CIM results, based on measurements across five chips.

$V_{\text{set}} = 1.2$ V applied from SL to BL, PW = 1 μs, VWL = 1.5 V, and target resistance value is <3 $k\Omega$; 3) RESET: $V_{\text{reset}} = 3.3$ V applied from BL to SL, PW = 100 ns, VWL = 3.3 V, and target resistance value is >50 $k\Omega$; and 4) READ: $V_{\text{read}} = 0.2$ V and VWL = 3.3 V.

In this work, as discussed earlier, we test and report the measurement results of the 1-bit/cell RRAM scheme as the intermediate resistance levels required for more than 1-bit/cell operation exhibit significant variance. Fig. 10(a) shows the measured RRAM LRS/HRS distribution across five test chips and the corresponding pattern match voltage distribution shown in Fig. 10(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 0.45 V and the VWL is 3.3 V. As we observed, a VSL voltage higher than 0.45 V may disturb RRAM resistance during inference operation. All the digital parts including SL/BL/WL decoders, counter/adder, SA, and scan-chains were powered up with 1.2 V for the experiment to generate Fig. 10(b). The highly parallel operations necessitate the activation of all the

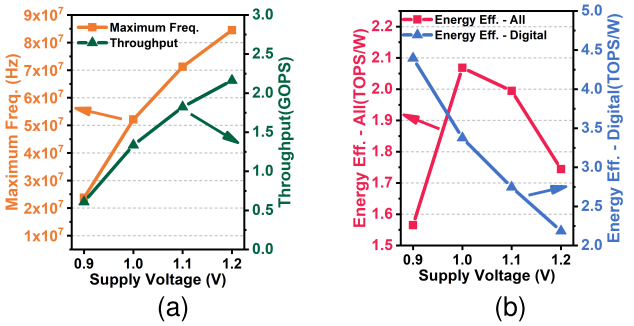
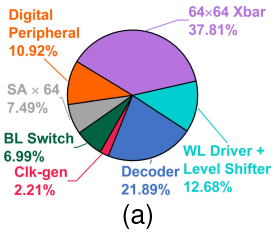


Fig. 11. Chip measurements with voltage scaling. (a) Frequency/throughput and (b) energy efficiency.



Design	This Work
Technology Node	65nm
RRAM Type	1T1R HfO_2
RRAM Array Size	64×64
1T1R Array 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)

Fig. 12. (a) Chip area breakdown and (b) chip specifications and performance summary.

SAs simultaneously, which may impact the static mismatch of the SAs. The experimental results reveal that the static mismatch follows a distribution with a mean of 1.9 mV and a standard deviation (sigma) of 14.07 mV. This distribution falls within the approximate 80-mV sensing margin [as shown in Fig. 10(b)], ensuring the correct XNOR result of the CIM macro.

The chip's core power consumption comes from two main sources: analog supply and digital supply. The analog supply feeds in from the SL through the given path of RRAM devices, with a fixed voltage 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 to 400 μW, 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, and digital driver for WL/SL/BL and SAs. The digital power strongly correlates with the supply voltage and operating frequency. We performed measurements with voltage scaling for the digital circuits from 0.9 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. 11. In Fig. 11(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 × 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 the RRAM array, which is 5 in this work for SAs and the parallel adder. At 1.2-V supply, we achieve the maximum frequency of 84.5 MHz and maximum throughput of 2.16 GOPS. As the supply voltage reduces, the frequency and throughput reduce largely linearly.

Fig. 11(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 II
ALIGNMENT COMPARISON WITH PRIOR RELATED WORKS

Metrics	CPU [26] AMD Opteron 6128	GPU [26] NVIDIA Tesla M2075	FPGA [28]	ASIC [26] CMOS	ASIC [27] CMOS	Ours CMOS+RRAM
Technology	45nm	40nm	28nm	40nm	28nm	65nm
Die Size (mm^2)	14.3k	1.6k	14.8	7.84	12	0.38
Power (W)	80	<200	247	0.135	0.975	0.01
Frequency (MHz)	2000	1150	200	200	400	84.5
Throughput (suffixes/s)	6.9×10^4	8.3×10^5	1.5×10^8	5.1×10^6	2.3×10^7	2.12×10^8
Energy Efficiency (suffixes/J)	870	4200	6.2×10^5	3.7×10^8	-	2.12×10^{10}
Throughput-to-Area (suffixes/s/mm^2)	200	1600	420	6.4×10^5	-	5.58×10^8

B. System-Level Performance Evaluation

In addition to the previously discussed chip measurement results and performance analysis (as summarize in Fig. 12), we have also carried out a system-level performance evaluation for both the applications.

1) Alignment-in-Memory: Table II shows the comparison of our chip prototype design with five different types of genome alignment platforms: CPU/GPU as general purpose processors, FPGA [28] implementation, and ASIC design [26], [27]. While CPUs/GPUs run at faster frequencies and have more ON-chip memory, the "memory wall" limits their absolute throughput and energy efficiency. An FPGA-based implementation achieves higher performance due to its large scale (eight FPGAs in this implementation) and dedicated dataflow graph. The related prior CMOS ASIC designs show 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 performance, 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.

2) Quantification-in-Memory: To the best of our knowledge, our chip design is unique in its application to mRNA quantification. For a fair comparison, we select SOTA NVM-based CIM designs that support high-parallelism XNOR operations as benchmarks. We adapt our CIM-friendly quantification-in-memory algorithm to these designs and extrapolate their original performance metrics to estimate their efficacy in mRNA quantification. The comparative results are detailed in Table III.

In our analysis, we assume a short-read length of 100 nts and a k-mer length of 12 nts, making these parameters compatible with all the considered CIM designs. Unlike our design, the competing designs, primarily developed for ML

TABLE III
QUANTIFICATION COMPARISON WITH PRIOR RELATED WORKS

	Ours	XNOR-RRAM [44]	Nature'21 [45]	VLSI'22 [46]
Memory	RRAM	RRAM	MRAM	MRAM
Technology	65nm	90nm	28nm	22nm
Num of Row.	64	128	64	256
Col. Parallelism	64	64	64	512
Throughput* (queries/s)	7.75×10^4	1.4×10^4	1.62×10^4	5.49×10^5
Energy Effi. (queries/J)	7.75×10^6	2.14×10^6	-	1.85×10^6
Throughput to Area (queries/s/mm2)	2.04×10^5	-	7.8×10^5	2.44×10^5

* Each query involves $(L-k+1) \times k$-times XNOR and $(L-k+1)$-time memory access for AND operation where L is the length of short-read, k is the length of k-mer.

applications, are configured to activate all the columns and rows simultaneously. This setup enables them to execute XNOR operations with all the saved data in a single step. However, these designs rely on ADCs or time-to-digital converters (TDCs) to count XNOR matches per column, a process necessary for identifying the matched k-mer index. The energy-intensive and time-consuming nature of ADC/TDC not only limits throughput but also requires a larger area. The TDC and ADC designs in [45] and [46] occupy 2.5–2.6× the area of the NVM array, while the SAs in our design occupy only around 20% of the NVM array's area. Specifically, the NVM array size in [46] is 256 × 512, which is 32× larger than our RRAM array. Moreover, in contrast to our design, which uses the simple 1T1R cell structures, the competing designs require more complex two-transistor-two-RAM or two-transistor-two-MTJ (2T2R/2T2M) cells for high-parallelism XNOR operations. These factors contribute to our design demonstrating performance metrics closely aligned with [45], [46], despite their fabrication using a significantly more advanced technology node.

VII. CONCLUSION

This work presents the first RRAM-based CIM macro tailored to advance the acceleration of genome processing. We delved into workflows of both the BWT-based genome short-read alignment and alignment-free mRNA quantification and identified the major required operations, including highly parallel **XNOR_*Match***, **AND**, and ***ADD***, which are seamlessly integrated into our fabricated 65-nm CIM macro chip. This innovative CIM macro design not only leverages the high parallelism of CIM architecture but also keeps the low complexity of the circuit, making complex genome processing tasks more efficient and less resource-intensive. Our experiments and measurements highlight the significant improvement in energy efficiency and throughput over SOTA different types of computing platforms, including CPUs, GPUs, previous CMOS ASIC designs, and prior CIM designs that use the same algorithms. This significant performance improvement offers the potential for a novel computing solution for energy-efficient, high-performance, and fast genome processing platform.

REFERENCES

[1] J. A. Reuter, D. V. Spacek, and M. P. Snyder, "High-throughput sequencing technologies," *Mol. Cell*, vol. 58, no. 4, pp. 586–597, May 2015.

[2] B. E. Slatko, A. F. Gardner, and F. M. Ausubel, "Overview of next-generation sequencing technologies," *Current Protocols Mol. Biol.*, vol. 122, no. 1, p. e59, 2018.

[3] Z. Wang, Z. He, M. Shah, T. Zhang, D. Fan, and W. Zhang, "Network-based multi-task learning models for biomarker selection and cancer outcome prediction," *Bioinformatics*, vol. 36, no. 6, pp. 1814–1822, Mar. 2020.

[4] H. Fecteau, *K*. J. Vogel, *K*. Hanson, and S. Morrill-Cornelius, "The evolution of cancer risk assessment in the era of next generation sequencing," *J. Genetic Counseling*, vol. 23, no. 4, pp. 633–639, Aug. 2014.

[5] M.-L. Frémond et al., "Next-generation sequencing for diagnosis and tailored therapy: A case report of astrovirus-associated progressive encephalitis," *J. Pediatric Infectious Diseases Soc.*, vol. 4, no. 3, pp. e53–e57, Sep. 2015.

[6] R. Mellis, N. Chandler, and L. S. Chitty, "Next-generation sequencing and the impact on prenatal diagnosis," *Expert Rev. Mol. Diagnostics*, vol. 18, no. 8, pp. 689–699, Aug. 2018.

[7] Y. Cho et al., "Prevalence of rare genetic variations and their implications in NGS-data interpretation," *Sci. Rep.*, vol. 7, no. 1, p. 9810, Aug. 2017.

[8] E. D. Pleasance et al., "A comprehensive catalogue of somatic mutations from a human cancer genome," *Nature*, vol. 463, no. 7278, pp. 191–196, 2010.

[9] M. A. Hamburg and F. S. Collins, "The path to personalized medicine," *New England J. Med.*, vol. 363, no. 4, pp. 301–304, 2010.

[10] T. F. Smith and M. S. Waterman, "Identification of common molecular subsequences," *J. Mol. Biol.*, vol. 147, no. 1, pp. 195–197, Mar. 1981.

[11] C.-H. Chang et al., "SBWT: Memory efficient implementation of the hardware-acceleration-friendly Schindler transform for the fast biological sequence mapping," *Bioinformatics*, vol. 32, no. 22, pp. 3498–3500, Jul. 2016.

[12] B. Langmead and S. L. Salzberg, "Fast gapped-read alignment with bowtie 2," *Nature Methods*, vol. 9, no. 4, pp. 357–359, Apr. 2012, doi: 10.1038/nmeth.1923.

[13] H. Li, "Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM," 2013, *arXiv:1303.3997*.

[14] H. Li and R. Durbin, "Fast and accurate long-read alignment with Burrows–Wheeler transform," *Bioinformatics*, vol. 26, no. 5, pp. 589–595, 2010.

[15] H. Li and R. Durbin, "Fast and accurate short read alignment with Burrows–Wheeler transform," *Bioinformatics*, vol. 25, no. 14, pp. 1754–1760, May 2009, doi: 10.1093/bioinformatics/btp324.

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

[17] X.-Q. Li, G.-M. Tan, and N.-H. Sun, "PIM-align: A processing-in-memory architecture for FM-index search algorithm," *J. Comput. Sci. Technol.*, vol. 36, no. 1, pp. 56–70, Jan. 2021, doi: 10.1007/s11390-020-0825-3.

[18] Y. Li, X. Li, R. Gao, W. Liu, and G. Tan, "NvWa: Enhancing sequence alignment accelerator throughput via hardware scheduling," in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit. (HPCA)*, Feb. 2023, pp. 1236–1248.

[19] F. Wen, M. Qin, P. Gratz, and N. Reddy, "OpenMem: Hardware/software cooperative management for mobile memory system," in *Proc. 58th ACM/IEEE Design Autom. Conf. (DAC)*, Dec. 2021, pp. 109–114.

[20] B. Li, P. Gu, Y. Shan, Y. Wang, Y. Chen, and H. Yang, "RRAM-based analog approximate computing," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 34, no. 12, pp. 1905–1917, Dec. 2015.

[21] A. Nag et al., "GenCache: Leveraging in-cache operators for efficient sequence alignment," in *Proc. 52nd Annu. IEEE/ACM Int. Symp. Microarchitecture*, Oct. 2019, pp. 334–346, doi: 10.1145/3352460.3358308.

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

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

[24] S. Angizi, J. Sun, W. Zhang, and D. Fan, "PIM-aligner: A processing-in-MRAM platform for biological sequence alignment," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2020, pp. 1265–1270.

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

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

[27] Y.-C. Wu et al., "A 975-mW fully integrated genetic variant discovery system-on-chip in 28 nm for next-generation sequencing," *IEEE J. Solid-State Circuits*, vol. 56, no. 1, pp. 123–135, Jan. 2021.
[28] J. Arram, T. Kaplan, W. Luk, and P. Jiang, "Leveraging FPGAs for accelerating short read alignment," *IEEE/ACM Trans. Comput. Biol. Bioinf.*, vol. 14, no. 3, pp. 668–677, May 2017.
[29] O. Mutlu and C. Firtina, "Accelerating genome analysis via algorithm-architecture co-design," in *Proc. 60th ACM/IEEE Design Automat. Conf. (DAC)*, Jul. 2023, pp. 1–4.
[30] F. Zhang, S. Angizi, J. Sun, W. Zhang, and D. Fan, "Aligner-D: Leveraging in-DRAM computing to accelerate DNA short read alignment," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 13, no. 1, pp. 332–343, Mar. 2023.
[31] M. Burrows and D. J. Wheeler, "A block-sorting lossless data compression algorithm," Digit. Equip. Corp., Maynard, MA, USA, Tech. Rep. 124, 1994.
[32] R. de Sousa Abreu, L. O. Penalva, E. M. Marcotte, and C. Vogel, "Global signatures of protein and mRNA expression levels," *Mol. BioSyst.*, vol. 5, no. 12, pp. 1512–1526, 2009.
[33] W. Zhang et al., "Network-based isoform quantification with RNA-seq data for cancer transcriptome analysis," *PLOS Comput. Biol.*, vol. 11, no. 12, Dec. 2015, Art. no. e1004465.
[34] N. L. Bray, H. Pimentel, P. Melsted, and L. Pachter, "Near-optimal probabilistic RNA-seq quantification," *Nature Biotechnol.*, vol. 34, no. 5, pp. 525–527, May 2016.
[35] M. Vasimuddin, S. Misra, H. Li, and S. Aluru, "Efficient architecture-aware acceleration of BWA-MEM for multicore systems," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, May 2019, pp. 314–324.
[36] 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, Feb. 2014.
[37] N. Ahmed, J. Lévy, S. Ren, H. Mushtaq, K. Bertels, and Z. Al-Ars, "GASAL2: A GPU accelerated sequence alignment library for high-throughput NGS data," *BMC Bioinf.*, vol. 20, no. 1, p. 520, Oct. 2019, doi: 10.1186/s12859-019-3086-9.
[38] Z. Bingöl, M. Alser, O. Mutlu, O. Ozturk, and C. Alkan, "GateKeeper-GPU: Fast and accurate pre-alignment filtering in short read mapping," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops (IPDPSW)*, Jun. 2021, p. 209.
[39] Y. Turakhia, G. Bejerano, and W. J. Dally, "Darwin: A genomics co-processor provides up to 15,000X acceleration on long read assembly," in *Proc. ACM Int. Conf. Architectural Support Program. Lang. Operating Syst. (ASPLOS)*, 2018, pp. 199–213.
[40] D. Fujiki et al., "GenAx: A genome sequencing accelerator," in *Proc. ACM/IEEE 45th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2018, pp. 69–82.
[41] Y. Turakhia, S. D. Goenka, G. Bejerano, and W. J. Dally, "Darwin-WGA: A co-processor provides increased sensitivity in whole genome alignments with high speedup," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2019, pp. 359–372.
[42] F. Zhang, S. Angizi, N. A. Fahmi, W. Zhang, and D. Fan, "PIM-quantifier: A processing-in-memory platform for mRNA quantification," in *Proc. 58th ACM/IEEE Design Automat. Conf. (DAC)*, Dec. 2021, pp. 43–48.
[43] J. Hazra, M. Liehr, K. Beckmann, M. Abedin, S. Rafq, and N. Cady, "Optimization of switching metrics for CMOS integrated HfO_2 based RRAM devices on 300 mm wafer platform," in *Proc. IEEE Int. Memory Workshop (IMW)*, May 2021, pp. 1–4.
[44] S. Yin, X. Sun, S. Yu, and J.-S. Seo, "High-throughput in-memory computing for binary deep neural networks with monolithically integrated RRAM and 90-nm CMOS," *IEEE Trans. Electron Devices*, vol. 67, no. 10, pp. 4185–4192, Oct. 2020.
[45] S. Jung et al., "A crossbar array of magnetoresistive memory devices for in-memory computing," *Nature*, vol. 601, no. 7892, pp. 211–216, Jan. 2022, doi: 10.1038/s41586-021-04196-6.
[46] P. Deaville, B. Zhang, and N. Verma, "A 22 nm 128-KB MRAM row/column-parallel in-memory computing macro with memory-resistance boosting and multi-column ADC readout," in *Proc. IEEE Symp. VLSI Technol. Circuits (VLSI Technology Circuits)*, Jun. 2022, pp. 268–269.

Fan Zhang (Student Member, IEEE) received the B.S. degree in electrical engineering from Tianjin Chengjian University, Tianjin, China, in 2015, and the M.S. degree in electrical engineering from the Stevens Institute of Technology, Hoboken, NJ, USA, in 2017. He is currently pursuing the Ph.D. degree in electrical and computer engineering, Johns Hopkins University, Baltimore, MD, USA.

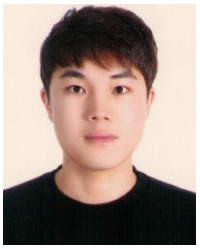
His research focuses on in-/near-memory computing circuits and architecture, software/hardware co-design, and machine learning algorithm acceleration.

Mr. Zhang is a recipient of the 2022 Best IP Award at the Design, Automation and Test in Europe Conference (DATE). More details are on his research website: https://www.xyzzhangfan.tech/

Amitesh Sridharan received the bachelor's degree in electrical engineering from the Sri Venkateswara College of Engineering, Chennai, India, and the master's degree in computer engineering from Arizona State University, Tempe, AZ, USA. He is currently pursuing the Ph.D. degree in Fall 2023 with the Johns Hopkins University, Baltimore, MD, USA.

His primary research interests are in-memory computing, AI hardware, and VLSI design.

Wangxin He received the Bachelor of Engineering degree in microelectronics from Xiamen University, Xiamen, China, in 2017, the Master of Engineering degree in electrical engineering from the University of Illinois at Chicago, Chicago, IL, USA, in 2019, and the Ph.D. degree in electrical engineering from Arizona State University, Tempe, AZ, USA, under Dr. Jae-Sun Seo by December 2023, specializing in NVM hardware acceleration design for artificial intelligence and machine learning research purpose.

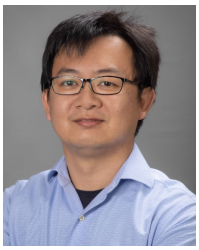


Injune Yeo (Member, IEEE) received the B.S. degree in semiconductor science from Dongguk University, Seoul, South Korea, in 2011, and the master's degree in mechatronics engineering and the Ph.D. degree in electrical engineering from Gwangju Institute of Science and Technology, Gwangju, South Korea, in 2014 and 2020, respectively.

From 2020 to 2022, he was a Post-Doctoral Scholar with the School of Electrical, Computer and Energy Engineering, Arizona State University, Tempe, AZ, USA. He is currently an Assistant Professor with the School of Electrical Engineering, Chosun University, Gwangju. His current research interests include an analog-to-digital converter, PUF, and in-memory computing.

Maximilian Liehr received the B.A. and M.Eng. degrees from the Rensselaer Polytechnic Institute, Troy, NY, USA, and the Ph.D. degree in nanoscale engineering from the College of Nanotechnology, Science, and Engineering, University of Albany, Albany, NY, USA.

He has active research interests in the development of non-volatile nanoelectronics including resistive random access memory (ReRAM) devices and neuromorphic computing.



Wei Zhang received the Ph.D. degree in computer science from the University of Minnesota Twin Cities, Minneapolis, MN, USA.

He is an Associate Professor and the Graduate Program Director in computer science at the University of Central Florida, Orlando, FL, USA. His research interests are broadly in cancer genomics, network-based learning in bioinformatics, and disease phenotype prediction.

Nathaniel Cady (Member, IEEE) received the B.A. and Ph.D. degrees from Cornell University, Ithaca, NY, USA.

He is currently an Empire Innovation Professor of Nanobioscience and an Associate Dean for Research at the University at Albany, Albany, NY, USA. He is also the Executive Director of the SUNY Applied Materials Research Institute (SAMRI), Albany, that funds collaborative research efforts between SUNY Faculty and Industry Partner Applied Materials (AMAT). He has active research interests in the development of novel biosensor technologies and biology-inspired nanoelectronics, including novel hardware for neuromorphic computing.

Yu Cao (Fellow, IEEE) received the B.S. degree in physics from Peking University, Beijing, China, in 1996, and the M.A. degree in biophysics and the Ph.D. degree in electrical engineering from the University of California at Berkeley, Berkeley, CA, USA, in 1999 and 2002, respectively.

He is a Professor with the Department of Electrical and Computer Engineering, University of Minnesota (UMN), Minneapolis, MN, USA. Before joining UMN, he was a Professor of electrical engineering at Arizona State University (ASU), Tempe, AZ, USA. He has published numerous articles and two books on nano-CMOS modeling and physical design. His research interests include neural-inspired computing, hardware design for ON-chip learning, and reliable integration of nanoelectronics.

Dr. Cao was a recipient of the 2022 Best IP Award at Design, Automation and Teste in Europe Conference, the 2022 Distinguished Lecturer of IEEE Circuits and Systems Society, the 2020 Intel Outstanding Researcher Award, the 2012 Best Paper Award at IEEE Computer Society Annual Symposium on VLSI, seven times Top 5% Teaching Awards at ASU, the 2009 ACM SIGDA Outstanding New Faculty Award, the 2009 Promotion and Tenure Faculty Exemplar at ASU, the 2009 Distinguished Lecturer of IEEE Circuits and Systems Society, the 2007 Best Paper Award at International Symposium on Low Power Electronics and Design, the 2006 NSF CAREER Award, the 2006 and 2007 IBM Faculty Award, the 2004 Best Paper Award at International Symposium on Quality Electronic Design, and the 2000 Beatrice Winner Award at International Solid-State Circuits Conference. He serves as the Specialty Chief Editor for Integrated Circuits and VLSI, Frontiers in Electronics. He was an Associate Editor of the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN and *Microelectronics Reliability*, Elsevier, and on the technical program committee of many conferences.

Jae-Sun Seo (Senior Member, IEEE) received the B.S. degree in electrical engineering from Seoul National University, Seoul, South Korea, in 2001, and the M.S. and Ph.D. degrees in electrical engineering from the University of Michigan, Ann Arbor, MI, USA, in 2006 and 2010, respectively.

From 2010 to 2013, he was with the IBM T. J. Watson Research Center, Yorktown Heights, NY, USA, where he worked on cognitive computing chips and high-performance IBM processors. From 2014 to 2023, he was an Assistant and Associate Professor with the School of Electrical, Computer and Energy Engineering, Arizona State University, Tempe, AZ, USA. Since 2023, he has been an Associate Professor with the School of Electrical and Computer Engineering, Cornell Tech, New York, NY, USA. He spent visiting researcher positions at the Intel Circuits Research Lab in 2015 and the Meta Reality Labs in 2022. His research interests include efficient ASIC/FPGA design of artificial intelligence (AI) and neuromorphic algorithms, and hardware-aware AI algorithm design.

Dr. Seo was a recipient of the Samsung Scholarship from 2004 to 2009, the IBM Outstanding Technical Achievement Award in 2012, the NSF CAREER Award in 2017, the Intel Outstanding Researcher Award in 2021, and IEEE TRANSACTIONS ON VERY LARGE-SCALE INTEGRATION Best Paper Award in 2022. He has served on the Technical Program Committee Member for ISSCC, ISCA, DATE, DAC, ICCAD, and MLSys. He serves as an Associate Editor-in-Chief for IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR ARTIFICIAL INTELLIGENCE, and an Associate Editor for the IEEE OPEN JOURNAL OF THE SOLID-STATE CIRCUITS SOCIETY, IEEE TRANSACTIONS ON BIOMEDICAL CIRCUITS AND SYSTEMS, and IEEE JOURNAL ON EXPLORATORY SOLID-STATE COMPUTATIONAL DEVICES AND CIRCUITS.

Deliang Fan (Member, IEEE) received the M.S. and Ph.D. degrees in electrical and computer engineering from Purdue University, West Lafayette, IN, USA, in 2012 and 2015, respectively.

He is currently an Associate Professor with the Department of Electrical and Computer Engineering, Johns Hopkins University, Baltimore, MD, USA. His primary research interests include machine learning circuits and algorithm at edge, in-memory computing circuits, and architecture, adversarial, and trustworthy AI system. He has authored and coauthored more than 170 peer-reviewed international journal/conference papers in those areas.

Dr. Fan is a recipient of the NSF Career Award, the Best IP Paper Award of DATE 2022, the Best Paper Award of GLSVLSI 2019, ISVLSI 2018, and ISVLSI 2017. His research works were also nominated as the Best Paper Candidate of ASPDAC 2019 and ISQED 2019. He is the Program Chair of ISQED 2024/2023, the Co-Chair of iMACAW@DAC from 2022 to 2024. He is also the Technical Area Chair of DAC 2021, GLSVLSI from 2019 to 2022, ISQED from 2019 to 2022, and the Financial Chair of ISVLSI 2019. More details are in his research website: https://www.ece.jhu.edu/dfan/