2023 IEEE 34th International Conference on Application-specific Systems, Architectures and Processors (ASAP)

SieveMem: A Computation-in-Memory Architecture for Fast and Accurate Pre-Alignment

Taha Shahroodi, Michael Miao, Mahdi Zahedi, Stephan Wong, Said Hamdioui
Dept. Quantum and Computer Engineering, Delft University of Technology, Delft, The Netherlands
Emails: {T.Shahroodi, M.Z.Zahedi, J.S.S.M.Wong, S.Hamdioui}@tudelft.nl, m.miao@student.tudelft.nl

***Abstract*—The high execution time of DNA sequence alignment negatively affects many genomic studies that rely on sequence alignment results. Pre-alignment filtering was introduced as a step before alignment to reduce the execution time of short-read sequence alignment greatly. With its success, i.e., achieving high accuracy and thus removing unnecessary alignments, the filtering itself now constitutes the larger portion of the execution time. A significant contributing factor entails the movement of sequences from the memory to the processing units, while a majority will filter out as they do not result in an acceptable alignment. State-of-the-art (SotA) pre-alignment filtering accelerators suffer from the same overhead for data movements. Furthermore, these accelerators lack support for future pre-alignment filtering algorithms using the same operations and underlying hardware. This paper addresses these shortcomings by introducing SieveMem. SieveMem is an architecture that exploits the Computation-in-Memory paradigm with memristive-based devices to support shared kernels of pre-alignment filters and algorithms inside the memory (i.e., preventing data movements). SieveMem architecture also provides support for future algorithms. SieveMem supports more than 47.6% of shared operations among all top 5 SotA filters. Moreover, SieveMem includes a hardware-friendly pre-alignment filtering algorithm called BandedKrait, inspired by a combination of mentioned kernels. Our evaluations show that SieveMem provides up to 331.1× and 446.8× improvement in the execution time of the two most-common kernels. Our evaluations also show that BandedKrait provides accuracy at the SotA level. Using BandedKrait on SieveMem, a design we call Mem-BandedKrait, one can improve the execution time of end-to-end sequence alignment irrespective of the dataset, which can go up to 91.4× compared to the SotA accelerator on GPU.**

***Index Terms*—Alignment, Pre-alignment Filter, Computation in Memory, Emerging Memory Technology, Hardware Accelerator**

I. Introduction

Sequence alignment[1] is a fundamental step in most genomic studies that help us with outbreaks surveillance, precision medicine, and other medical advances [11, 21, 36]. Sequence alignment is finding the similarity/closeness between a reference genome sequence (hereafter called reference) and a DNA read sequence (hereafter called read). Unfortunately, the DNA base-pairs (e.g., A, C, G, T) in references and reads may not always be identical at the location the read actually comes from for two reasons: (1) errors that arise when obtaining the sequences (a process called genome sequencing [32, 42]), and (2) genetic differences that exist among an individual organisms' DNA and corresponding reference [3]. Therefore, the sequence alignment process should be able to tolerate such differences, commonly known as edits: deletion, insertion, or substitution. To deal with this requirement, SotA sequence alignment methods employ computationally costly dynamic programming-based (DP) algorithms such as Needleman-Wunsch or Smith-Waterman algorithms [14, 34, 45] to account for edits while avoiding duplicate works. Unfortunately, these DP algorithms are computationally costly and incur long latencies and energy inefficiencies when applied to large DNA sequences. These limitations directly affect the medical studies that benefit from sequence alignment.

Pre-alignment filtering[2] was recently introduced as a solution to significantly speed up the overall process of sequence alignment by heuristically replacing the need for expensive DP solutions for many inputs, given a pre-defined edit distance threshold between the inputs [2, 4, 5]. SotA pre-alignment filters speed up the short-read (100-250 base-pairs or bps) sequence alignment so much so that they themselves become the (next) bottleneck to be accelerated [2, 4]. Although one SotA work [5] accelerates the pre-alignment filters on Graphics Processing Units (GPUs) and Field-Programmable Gate Arrays (FPGAs), this work still does not completely alleviate the bottleneck. Moreover, we find that data movement is a major issue in SotA pre-alignment filters, i.e., these filters waste a lot of time and energy when moving the sequences from the memory to processing units, most of which turn out to be unnecessary as the data is decided to be filtered out eventually [5]. Therefore, there is a need for a more efficient design to tackle the filtering bottleneck in the sequence alignment pipeline and simultaneously avoid the wasted work, time, and energy consumption caused by data movement in the system.

We propose SieveMem, an architecture based on Computation-in-Memory (CIM) principles capable of handling shared kernels in pre-alignment filtering for short-sequence alignment. We identify the shared kernels via an extensive profiling process using the same datasets and platforms for all the pre-alignment filters for a fair comparison and accurate recognition of bottlenecked operations. We also propose BandedKrait, a novel lightweight pre-alignment filter from shared kernels in previous filters (i.e., those supported by SieveMem) and its mapping into SieveMem architecture, a design called Mem-BandedKrait. SieveMem adapts the CIM paradigm since it requires prevention of data movement,

[1] Also known as mapping

[2] We use the term filter and pre-alignment filter interchangeably hereafter.

DOI 10.1109/ASAP57973.2023.00035

processing a large amount of data, and performing relatively small and/or simple computations, the main characteristics a CIM architecture embraces [6, 37, 57]. SieveMem's design comprises two abstraction levels: (1) A low-level abstraction that supports the shared kernels in filters and (2) a high-level abstraction that supports filtering algorithms using the existing hardware in SieveMem and takes care of input data distribution and output data processing. SieveMem is designed to support filtering for short reads because most of the available data in the genomics realm is still short reads (sequences of length 100 to 250 base-pairs), even though the industry is slowly moving towards long-read sequencing. Therefore, supporting short reads filtering and alignment will remain relevant problems in upcoming years.

Our results show that SieveMem accelerates the execution time of the identified shared kernels by up to 331.1× and 446.8× for the two most common kernels in pre-alignment filters. The results also show that BandedKrait achieves an accuracy on par with SotA filter SneakySnake. When accelerated on SieveMem, Mem-BandedKrait accelerates pre-alignment filtering by up to 95.5× and 1292× over SotA filters on GPU and CPU, respectively, for the same real input datasets. SieveMem achieves all these benefits, neither replacing the sequence alignment nor introducing extra false negatives into the pre-alignment filtering process, i.e., SieveMem only affects the pre-alignment filtering step positively. Therefore, users can still employ SieveMem with any sequence aligner.

Our contributions are the following:

- An configurable memristor-based accelerator (called SieveMem) that supports the most common shared kernels in pre-alignment filters.
- A memory-friendly filtering algorithm, called BandedKrait, accounting for the inflexibility of having the starting point access aligned with the memory units in memory/hardware. BandedKrait uses the same shared kernels and corresponding hardware of previous filters.
- A CIM-enabled realization of BandedKrait on SieveMem, called Mem-BandedKrait, for short-read pre-alignment filtering.
- An extensive evaluation of SieveMem's supported kernels, BandedKrait, and Mem-BandedKrait using real data against previous software and hardware pre-alignment filters.

II. Background

In this section, we briefly discuss the necessary background for this work. We refer the readers to comprehensive reviews on the same topics [3, 22, 35] for more details.

A. Sequence Alignment

Sequence alignment is defined as identifying the potential matching locations of every genome sequence (called read) with respect to another known genome sequence, such as a representative sequence for a species (known as the reference genome). SotA sequence aligners use computationally costly DP algorithms to prevent unnecessary, duplicate work. There are two main directions to improve the sequencing alignment step directly [5, 12, 15, 25, 31]: 1) accelerating the DP algorithms, 2) exploiting the inherent parallelism of algorithms and accelerating them using high-performance computing platforms such as CPUs, FPGAs, and GPUs.

B. Pre-Alignment Filtering

Pre-alignment filtering is a heuristic-based method to mitigate the cost of sequence alignment by quickly eliminating the need for performing the expensive DP given a predefined threshold called "edit distance." SneakySnake [5], Shouji [2], MAGNET [4], and SHD [54] are a few widely-used examples of such filters. SneakySnake [5] is the most recent of such filtering techniques that proposes to reduce the approximate string matching (ASM) problem to the single net routing (SNR) problem to find the optimal path with the least routing cost. This tweak enables SneakySnake to filter most unnecessary alignments in a parallel and highly accurate manner. Alser, et al. [5] show that this conversion also makes SneakySnake suitable for other high-performance computing (HPC) architectures, e.g., GPUs.

C. Computation-in-Memory (CIM) and Memristors

The recently re-ignited Computation-in-Memory (CIM)[3] paradigm is a promising way to eliminate data movement and saves us time and energy as the bandwidth is the largest near the memory. This paradigm advocates for computation where the data resides and, therefore, effectively mitigates the need for data transfer. This contrasts with the traditional Von Neumann architecture in which data and processing units are separate and data movement is required for any operation. Previous works from industry and academia propose architectures based on this paradigm to improve the performance and energy consumption in applications with relatively small and/or simple computations and work on large amounts of data, such as those in Machine Learning and Bioinformatics [6, 10, 41]. Pre-alignment filtering algorithms enjoy the same properties.

A memristive device is a non-volatile emerging memory technology that stores the data through its resistance level [26, 51]. PCM [26], STT-RAM [50], and ReRAM [51] are just a few examples. Memristive devices are suitable candidates for both storage and computation units. Recent works combine the CIM paradigm with memristors and use them in crossbar-based memory structures to perform matrix-vector [52] and bulk bit-wise logical [53] operations efficiently following Kirchhoff's law. Memristor devices also enjoy high integration density and near-zero standby powers. Due to these features and our accessibility to accurate models and chip measurements for memristor-based memories (see Section V-A), we use memristors as our underlying technology in SieveMem.

III. Motivation and Profiling

This section (1) identifies the shared kernels in filters and (2) motivates the CIM-enabled acceleration of filters. We refer to Section V-A for detail on our setup and datasets.

[3]Interchangeably also known as Processing-in-Memory (PIM).

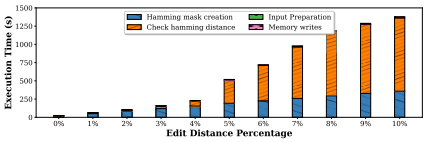

Fig. 1: MAGNET's breakdown.

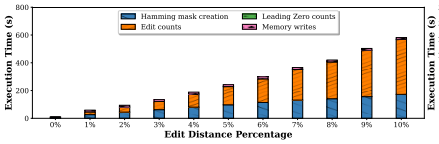

Fig. 2: Shouji's breakdown.

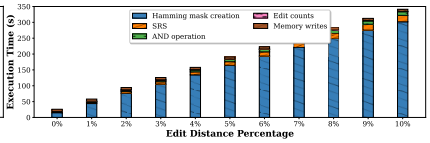

Fig. 3: SHD's breakdown.

A. Shared Kernels in Filters

We profile three SotA (i.e., accurate and fast) pre-alignment filters over different percentages of edit distances. Fig. 1, Fig. 2, and Fig. 3 present a breakdown of the execution time for each non-overlapping kernel in MAGNET, Shouji, and SHD, respectively, over our representative dataset.

We make three key observations. First, across all filters and edit distances, two operations, namely Hamming mask creation (HMC) and detecting short patterns (called Short Pattern Detect or SPD), make up most of the execution time. Note that we categorized checking hamming distance, leading zero counts, and SRS, in MAGNET, Shouji, and SHD, respectively, into SPD, as they are all essentially detecting short patterns. For example, HMC accounts for up to 66%, 84%, and 88% in end-to-end execution time of MAGNET, Shouji, and SHD, respectively. Moreover, up to 72% and 68% of the end-to-end execution time in MAGNET and Shouji, respectively, is spent on SPD.

Second, the relative percentage of these two operations varies per filter, edit distance, and dataset (and, therefore, is data dependent). However, the combined HMC and SPD account for a minimum 47.6% of filter's execution time, going up to 97.5%.

Third, apart from HMC and SPD, these filters also share other operations of the same nature. For example, kernels for counting edits, pre- and post-processing inputs and outputs, and extra reads and writes of intermediate results.

We conclude that HMC and SPD comprise most of the execution time in SotA filters. Acceleration of these two kernels can resolve the bottleneck in filters and simultaneously provide some assurance for future compatibility of filters that utilize the same kernels.

B. Data Movement in Fitlers

Fig. 4 presents the breakdown of execution time for a SotA accelerated filter, SneakySnake on GPU (Snake-on-GPU), on two very different datasets of short reads labeled as D1 and D2.

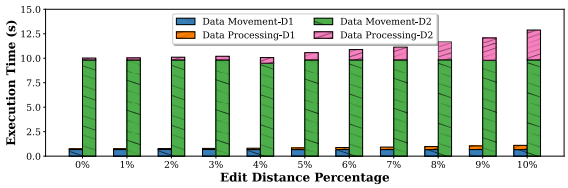

Fig. 4: Data movement bottleneck in accelerated filters.

We observe that over both datasets, Snake-on-GPU spends a minimum of 60% of its execution time just transferring data from memory to GPU. This portion can go up to 98%, depending on the dataset and edit threshold. We conclude that data movement is the bottleneck of the overall performance in SotA accelerators for filtering.

The results presented in this section call for an acceleration of pre-alignment filters with an emphasis on eliminating data movement and supporting commonly shared kernels such as HMC and SPD.

IV. Proposal and Architecture

This section discusses (1) SieveMem and how it supports HMC and SPD and mitigate data movement, (2) BandedKrait, our new lightweight pre-alignment filtering algorithm, and (3) Mem-BandedKrait, hardware realization of BandedKrait on SieveMem.

A. SieveMem Architecture

Fig. 5-(a) presents the placement of SieveMem in a real system, i.e., part of the memory. Due to this placement, SieveMem follows a hierarchical structure similar to conventional memories, i.e., SieveMem consists of ranks, bank groups, banks, subarrays, and tiles (Fig. 5-(b), -(c), and -(d)).

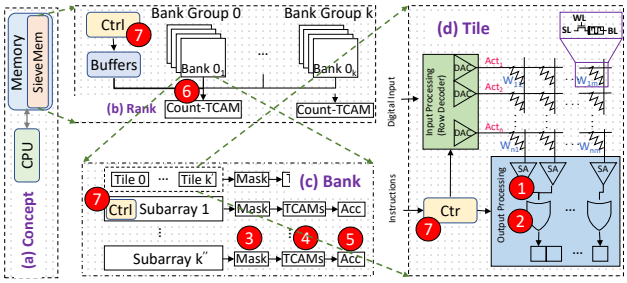

Fig. 5: (a) SieveMem system placement, (b) to (d) An overview of SieveMem hierarchy and its additional components at different levels.

However, to support the target shared kernels for filtering, SieveMem augments the substrate as we will discuss below:

Tile level changes: SieveMem enhances the SAs (1) and adds a series of OR gates (2). The modified SAs [28, 53] enable SieveMem to perform logical operations such as XOR, AND, etc., with a minimal area overhead on data in an entire row of a tile. This design is possible due to the nature of memristor devices that inherently follow Kirschof's law. The subsequent series of OR gates relates to the nature of our working datasets and encoding. We want SieveMem to enable filtering for DNA

short-reads of {A, C, G, T}. This means we can encode each character with 2 bits in a hardware realization. Since our enhanced SAs perform bitwise operations, SieveMem needs this series of OR to obtain a result based on base-pairs. Our walk-through example in Section IV-B details this further.

Bank level changes: SieveMem adds a series of AND gates for masking ❸, 2 TCAMs ❹, and a series of AND gates ❺. SieveMem uses AND gates to select any section of the outputs from tiles. The masking gates are necessary for a true CIM-enabled design where we cannot guarantee our target data is aligned perfectly with crossbars. A limitation that previous CIM-enabled designs typically face [16]. SieveMem uses the two memristor-based TCAMs (called Pattern-detect and Output-select) for all the necessary pattern detection in different kernels of pre-alignment filters. The AND gates are used to accumulate the results over several related checks in the SieveMem, for example, checks for the same read sequence over shifted versions of the reference. Section IV-B details how one can use different masks, pre-filled TCAMs, and bitwise gates to perform different operations and pre-alignment filtering algorithms.

Rank level changes: SieveMem includes a TCAM called Count-TCAM (❻) at the rank level per each bank group. SieveMem uses this TCAM to effectively calculate the minimum edit between the sequences.

Overall modifications: SieveMem adds a small FSM to control each hierarchy level's logic and memory operations (❼). SieveMem also includes some input and output buffers in different levels. Each level's FSM oversees the operations of the components in that level of SieveMem. These controllers are hierarchical, i.e., smaller FSMs control the operations at bank, subarray, and tile levels, all managed by a controller at the higher level. The buffers ensure seamless dataflow among different levels with no data loss.

B. SieveMem Example Support for SHD

SHD requires removing sequences of 1 to 2 zeros from the bit vector produced by the tile after the similarity checks via XOR. The original algorithm [54] performs this by detecting patterns of "101" and "1001". Here, SieveMem opts for an inverse detection, i.e., SieveMem detects sequences of zeros that are 3-bits or longer. Therefore, the output vector should be the same as the input vector but with the removal of the short '0' sequences.

SieveMem provides support for this via the TCAMs at the bank level. To implement the required pattern detection and selection, the output should always be a '1', except for those bits which are part of a sequence of 3 zeros or more. This is the case when a pattern of **0**00, 0**0**0 or 00**0** is detected, where the bold character indicates the position of the bit in the original bit vector at the input of the Pattern-detect TCAM. Therefore, the Output-select TCAM essentially performs a NOR operation on the Pattern-detect output corresponding to those 3 patterns. If any of the three patterns is detected, it will result in a '1' in the intermediate signal. The Output-select TCAM will, therefore, not output a '1' but a '0'. Conversely, if none of the patterns is detected, then SieveMem can be sure that the bit is not part of a sequence of 3 or more zeros. The intermediate result will contain only 3 zeros for these patterns, and the Output-select TCAM will output a '1' for that bit position in the XOR result.

Since the 2 base-pairs to the left and rightmost extremities of the TCAM input require information about the base-pairs to the left and right, respectively, these positions will always be left '0', done by the left and rightmost columns of the Output-select TCAM. Since the output of the bottom two rows of the Pattern-detect TCAM is always '1' due to it being filled with don't-cares, the output of the Output-select TCAM, which looks for '0', will always be '0'.

Fig. 6 presents how one can pre-fill the two TCAMs of our SieveMem to support the detection pattern required by SHD. Note that the blank TCAM entries stand for don't-cares. Here, there is a '0' between the two '1's in the 6th bit counting from the left. We observe that the patterns surrounding this bit are "010", "101", and "010". None of these are "000". Therefore, the Output-select TCAM will activate the row in green.

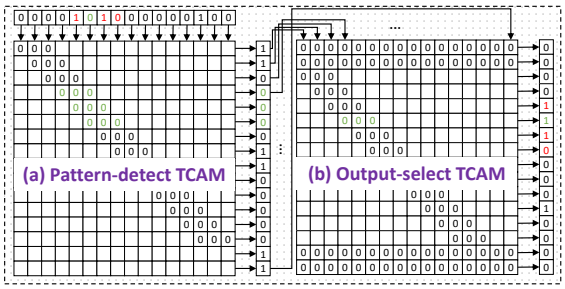

Fig. 6: TCAMs values in SieveMem to support SHD.

At the rank level, SieveMem collects all bank group results and counts the number of edits in a word set of the read sequence. SieveMem uses Count-TCAM to detect patterns and to assign a number for the edits of the detected patterns. Count-TCAM is a 4-bit wide TCAM used in algorithms where we need to split the final bit-vector into segments of k bits, e.g., SHD with k=4. Fig. 7 presents an example programming for Count-TCAM to support SHD.

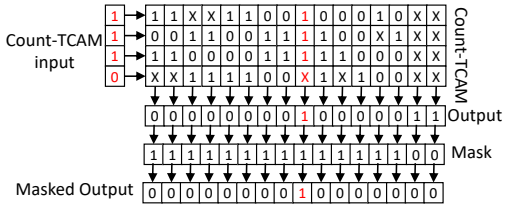

Fig. 7: Filled Count-TCAM for SHD.

For the segment of "0000", SHD counts no edits, while it counts two edits for "0101", "0110", "1001", "1010", "1011" and, "1101". The rest of the cases are counted as single edits. Therefore, SieveMem compresses these into 14 entries with

don't care cells, where the patterns that count for two edits have double entries. Note that SieveMem relays the inputs to the output without additional mutations. Therefore, a mask might be required for the output to ignore the unused entries.

C. BandedKrait Algorithm

SieveMem is capable of supporting simple, shared kernels in pre-alignment filters. Therefore, SieveMem can support any future algorithm that uses similar kernels with different control sequences. To show this, we devise a simple algorithm called BandedKrait[4,5].

BandedKrait reduces costly DP problem to a simpler exact matching problem between (shifted versions of) smaller segments of read and reference. To this end, BandedKrait divides the read sequence into segments of k-bps and compares them to the corresponding segments of the reference and its shifted versions. BandedKrait checks each pair for an exact match, and the results determine whether an edit is present within that segment. The repetition of this process for references shifted by $-E$ to $+E$ ensures that BandedKrait support up to E deletions and/or insertions. BandedKrait uses the pigeon-hole principle on the combined results of segments to approximate the #edits in the sequence pairing. Algorithm 1 summarizes BandedKrait, where k is the segment size and E is the number of permissible edits between input read and the reference sequence.

Algorithm 1 BandedKrait Algorithm

Input: Read, Reference, E, ReadLength, k
Output: Accept
$N_{segment} \leftarrow \lceil ReadLength/k \rceil$
$Matches \leftarrow 0$
for $i \in \{0 : N_{segment} - 1\}$ **do**
  $Match \leftarrow 0$
  for $e \in \{-E : +E\}$ **do**
    $ReadSegment \leftarrow Read[i \times k : (i+1) \times k - 1]$
    $ReferenceSegment \leftarrow Ref[i \times k + e : (i+1) \times k - 1 + e]$
    if $ReadSegment == ReferenceSegment$ **then**
      $Match \leftarrow 1$
    end if
  end for
  $Matches \leftarrow Matches + Match$
end for
$Accept \leftarrow (Matches >= N_{segment} - E)$
return $Accept$

BandedKrait flexibly explores two trade-offs: (1) accuracy vs. hardware-friendliness and (2) required resources vs. achievable parallelism or performance. Exact matching is known to be well-supported in hardware and specifically in a CIM-enabled crossbar. However, using exact matches as proximity to existing errors (compared to alternatives such as DP or SNR sub-problems in SneakySnake, for example) underestimates the number of edits. Therefore, more reads can pass BandedKrait, making it inaccurate. Moreover, finding the exact matches between each segment pair and the shifted variants are independent, parallelizable problems. However, exploiting that demands higher resources.

D. BandedKrait on SieveMem (Mem-BandedKrait)

BandedKrait is completely supported by SieveMem. If implemented on SieveMem, we call the design Mem-BandedKrait. Fig. 8 presents an example of how the two TCAMs in SieveMem are filled so that it can support the pattern required by BandedKrait algorithm. Mem-BandedKrait repeats this process for all $2E+1$ shifted reference segments. If no exact match is detected in any of the iterations, Mem-BandedKrait counts an error for that segment.

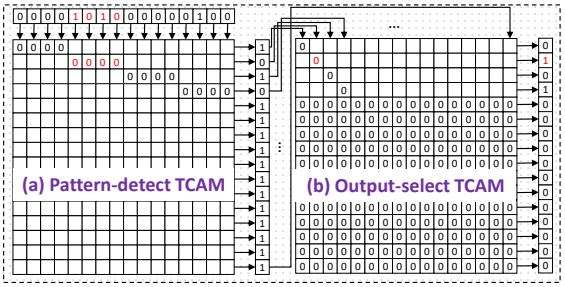

Fig. 8: TCAMs values in SieveMem to support BandedKrait.

V. Evaluations

A. Evaluation Methodology

Implementation & setup. We implemented SieveMem in a cycle-accurate RTL-based simulation platform. We verify the design by comparing the simulation results with SieveMem's software outputs. We use the same implementation for the evaluation of Mem-BandedKrait. SieveMem hardware uses memory models from a small RRAM crossbar in TSMC 40 nm CMOS technology [24, 43]. These memories are provided to us by generous partners from the EU project MNEMOSENE [33]. The additional components of SieveMem discussed in Section IV-A are also designed using TSMC 40 nm technology node in Synopsis Design Compiler [47]. We integrated the latency numbers into the simulation platform. We run our experiments on a 12-core server with 16 GB memory, Tesla-K80 GPUs, and a Intel® Xeon® CPU E5-2680 operating at 2.4 GHz. Our evaluations consider the same platform and input datasets for all filters for a fair analysis.
Baselines. We compare different kernels supported in SieveMem with their accelerated version in literature. We also compare Mem-BandedKrait and BandedKrait with SneakySnake (SS) [5], Shouji [2], SHD [54], and GRIM-Filter [23], as SotA pre-alignment filters. We use open-sourced implementations of these filters. We consider SneakySnake on both CPU and GPU, Shouji and SHD on FPGA, and GRIM-Filter on 3D-stacked memories.
Datasets. We use real genome datasets, i.e., $human_g1k_v38$. Since SieveMem is designed for supporting filters (and their

[4]The banded krait (Bungarus fasciatus) is a species of elapid snake easily identified by its alternate black and yellow crossbands, all of which encircle the body.

[5]Recently, Shahroodi et al. [39] propose RattlesnakeJake, a hardware/software (HW/SW) co-designed accelerator based on Computation-in-Memory (CIM) paradigm, capable of pre-alignment filtering for short-sequence alignment. The software algorithm behind RattlesnakeJake is similar to BandedKrait. However, the hardware design of RattlesnakeJake and SieveMem differ.

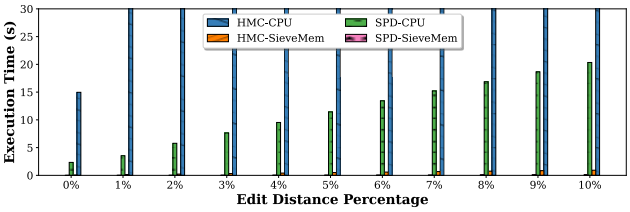

Fig. 9: HMC and SPD on ERR240727_1 with E=40.

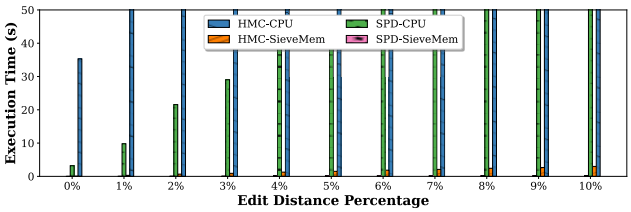

Fig. 10: HMC and SPD on SRR826471_1 with E=100.

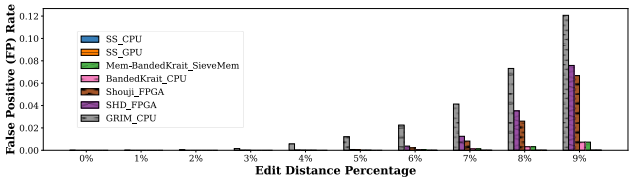

Fig. 11: FP rate on ERR240727_1 with E=40.

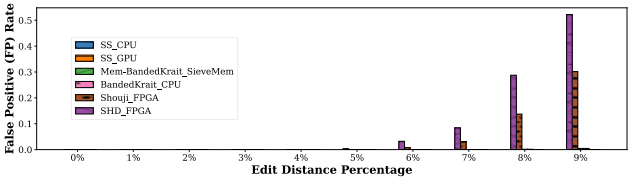

Fig. 12: FP rate on SRR826471_1 with E=100.

kernels) for short-reads, we use two sample sets [5] of $ERR240727_1$ and $SRR826471_1$ from Illumina reads [8] with read lengths varying from 100bps to 250bps, respectively. For end-to-end evaluation of alignment (in the case of comparing Mem-BandedKrait with other filters), we use Edlib [46] to create full-alignment results for accuracy. Edlib results also verify the functionality of filters.

B. Execution Time of Supported Kernels

Fig. 9 and Fig. 10 compare the execution time of performing the same number of HMC and SPD operations on CPU and SieveMem, over our two datasets for different edit distances. We choose SHD as the reference filter to align with the example we provided over SieveMem supporting SHD in Section IV-B. The results consider the necessary data movement. The y-axis has been limited to a low number (30 and 50) for improving readability. Since no approximation is used for these kernels, no accuracy loss occurs due to the underlying platform, and all three versions produce the same result.

We observe that SieveMem accelerates both HMC and SPD irrespective of the dataset. This improvement goes up to 331.1× and 446.8× for HMC and SPD, respectively, over the datasets. This is expected for two reasons: (1) fewer data movement in SieveMem and (2) parallel computation of target operation with a reasonably high clock cycle. We conclude that not only SieveMem supports the shared kernels of pre-alignment filters, but also it significantly accelerates them.

C. Filtering Accuracy

Fig. 11 and Fig. 12 compare the false positive (FP) rate of several filters. FP rate in a filter shows the ratio between reads that wrongly pass the filter (i.e., could have been filtered) and go through alignment (i.e., DP) over all the reads. The lower the FP, the better. Note that in terms of True Positive (TP) and True Negative (TN) rates (the other two important metrics for the accuracy of a filter), BandedKrait and Mem-BandedKrait achieve the same rate as SneakySnake, which currently results in the best filtering rates.

We make three key observations. First, BandedKrait and Mem-BandedKrait provide low FP rates irrespective of edit threshold and dataset. In fact, their FP rates are on par with the SotA SneakySnake. Second, the FP rate of Mem-BandedKrait and BandedKrait is less than 1% apart; i.e., the hardware limitation regarding the reference's start point, which affects #segments, does not affect the accuracy significantly. Third, BandedKrait outperforms Shouji, SHD, and GRIM-Filter by providing, on average, 22%, 40%, and 90%, respectively, fewer number of falsely-accepted sequences. We conclude that BandedKrait is an effective and accurate filter on both CPU and SieveMem architecture.

D. Filtering Speed

Fig. 13 and Fig. 14 present the execution time for different filtering methods over different edit threshold. The y-axis uses a logarithmic scale.

We make two key observations. First, although BandedKrait requires more time than some SotA filters, Mem-BandedKrait significantly outperforms the fastest SotA filters on CPU (SS_CPU) and on GPU (SS_GPU) by up to 1292× and 95.5×, respectively, when processing the same amount of sequences. Note that SS_CPU and SS_GPU outperform all previous existing filters, independent of the dataset and edit threshold.

Second, Mem-BandedKrait provides better scalability for larger short reads (SRR826471 vs. ERR240727) than other methods. For example, the average speedup of Mem-BandedKrait over SS_CPU is 2.93× more on SRR826471 than on ERR240727 for the same edit threshold of E=5. This is because the performance on Mem-BandedKrait is only slightly affected by the length of inputs and target edit threshold, while, in original SS_CPU this change is more significant.

We conclude that BandedKrait and Mem-BandedKrait effectively reduce the execution time of filtering for the same #processed read and reference sequences.

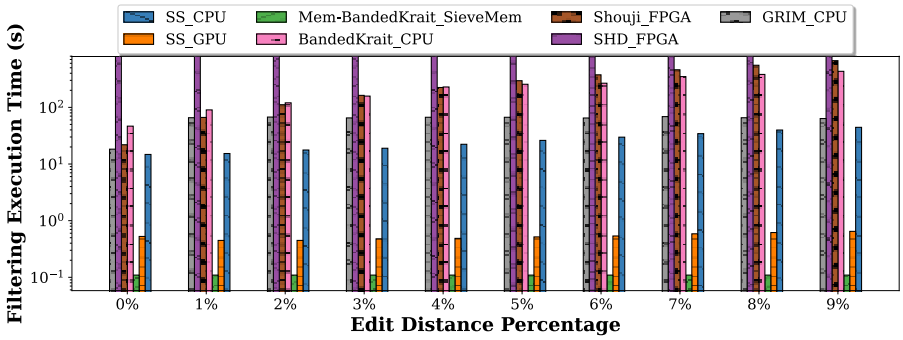

Fig. 13: Filtering speed on ERR240727_1 with E=40.

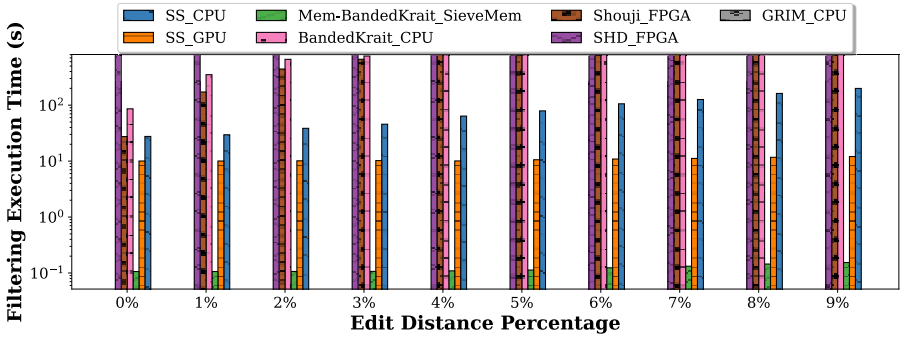

Fig. 14: Filtering speed on SRR826471_1 with E=100.

E. End-to-end Alignment Speed

Fig. 15 and Fig. 16 present the execution time for end-to-end alignment for a combination of filters with Edlib for alignment over several edit thresholds. We limit the y-axis that shows the execution time of filter+alignmnet to 1000s to capture better the trends and relative execution time of in the system compared to other methods. The y-axis is in logarithmic scale.

We make two key observations. First, Mem-BandedKrait significantly reduces the end-to-end execution time of sequence alignment irrespective of the dataset or edit threshold. The improvements are so profound that they are hard to capture, even on the logarithmic scale. Particularly, the improvement in filtering translates to a 254.6× and 91.4× improvement in end-to-end alignment time compared with SotA filter combined with SotA alignment and sheer Edlib on CPU, respectively, averaged over our datasets.

Second, similar to the comparison of filters in Section V-D, the speedup in the end-to-end alignment is higher for SRR826471 compared to that on ERR240727. For example, the average speedup of alignment using Mem-BandedKrait is 9.86× more on SRR826471 compared to on ERR240727 for edit threshold of E=5. This is due to the effect of filtering being even more effective on the SRR826471 dataset compared to the ERR240727 dataset as discussed in Section V-D.

We conclude that Mem-BandedKrait is a fast pre-alignment filter and effectively reduces the execution time of end-to-end alignment such that it takes a step towards mitigating the filtering bottleneck.

VI. Discussions and Future Works

A. SieveMem for Long Sequence Alignment

From the conceptual point of view, the BandedKrait algorithm is also effective for long sequence alignment, where sequences are a size of 100Kbp. However, when it comes to mapping to SieveMem, distributing the long reference or read sequences in the memory hierarchies requires complex bookkeeping, different buffer sizes, control unit sequences, and potentially some additional logic. However, to the best of our knowledge, pre-alignment filters are not currently deployed for long-read sequence alignment acceleration. We leave the exploration of BandedKrait on SieveMem for long pre-alignment filtering to future work.

B. Potential Design Explorations

SieveMem's best configuration. Current evaluations of SieveMem are based on the measurements on a small ReRAM chip prototype for tiles and TCAMs. However, a complete design space exploration is required to direct the final configuration of SieveMem before deploying it in future genomics systems. Such exploration should consider different inputs, memory units and arrangements, variations of devices, and circuit behavior (e.g., non-idealities) for different organizations. We leave this to an extended report.

Other memory technologies. We design SieveMem assuming memristors as the underlying technology due to the benefits they offer in terms of density, low power, and support

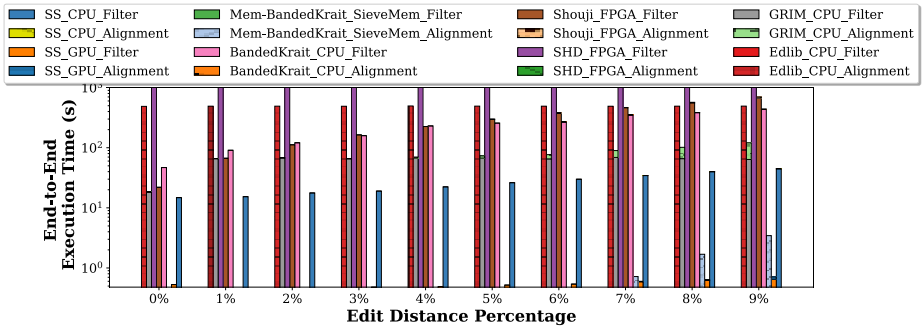

Fig. 15: End-to-End speed on ERR240727_1 with E=40.

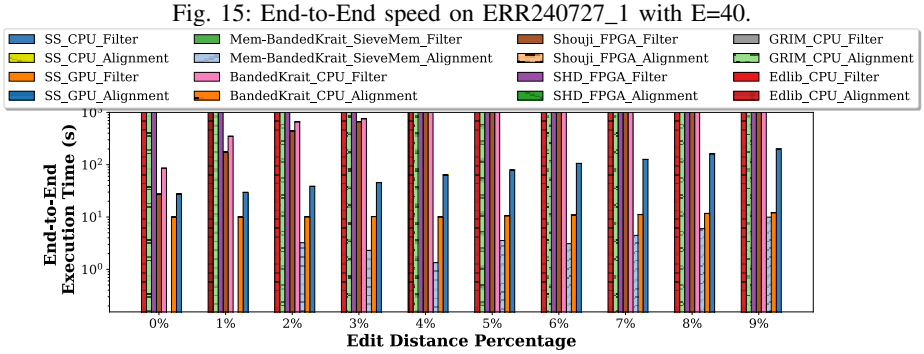

Fig. 16: End-to-End speed on SRR826471_1 with E=100.

for logical vector operations (please see Section II-C for more details). However, independent and separate works [37, 49, 55] propose supporting the same operations (XNOR, associate search, etc.) in other technologies as well. Having a complete comparison of SieveMem's versions with different technologies is an interesting work we leave for the future.

VII. Related Works

To our knowledge, this is the first work to present a CIM architecture to support shared kernels in pre-alignment filters. Our Mem-BandedKrait is also the first hardware filter that resolves the execution bottleneck of the filtering and alignment process (Section I and Section II). We have already extensively compared SieveMem, BandedKrait, Mem-BandedKrait with previous pre-alignment filters and their accelerators in Section V. This section briefly discusses previous works on sequence alignment and (PIM-enabled) genomics accelerators.

Sequence Alignment Acceleration. Many previous works aim to accelerate sequence alignment following two main directions: 1) accelerating the dynamic programming algorithms [12, 25], 2) exploiting the inherent parallelism of algorithms and accelerating them using high-performance computing platforms such as CPUs [13, 20], FPGAs [7, 15], and GPUs [29, 31]. SieveMem and Mem-BandedKrait (and in general all pre-alignment filters) are orthogonal to these works as they are pre-alignment filters and accelerators aiming to speed up the overall sequence alignment process by preventing the expensive DP in the first place.

General CIM-based Accelerators. Prior works heavily investigate different compute-capable memories [1, 16, 27, 37, 44, 58]. These works focus on different memory technologies (e.g., DRAM [19] vs. STT-MRAM [44]), various operations (e.g., simple logical operations [37] vs. more complex Vector-Matrix-Multiplication operation [38]), to even diverse applications (e.g., Binary Neural Network [38] vs. graph processing [56]). SieveMem differs from these works in either underlying technology, supported kernels, and/or accelerated application.

(CIM-based) Genomics Accelerators. Many works propose different architectures and accelerators for genomics-related kernels and applications. GenASM [9] and DARWIN [48] target approximate string matching and sequence alignment. ApHMM [18] uses hardware/software co-designed to accelerate the Baum-Welch algorithms used in pHMM graphs. Helix [30] and KrakenOnMem [41] accelerate basecalling and metagenomics profiling on memristor-based systems. BLEND [17] proposes an efficient mechanism to identify exact-matching and highly similar seeds through a single lookup of their hash values. BLEND uses a technique called SimHash and demonstrates its effectiveness in read overlapping and read mapping. Demeter [40] proposes a CIM-enabled architecture and a PCM-based accelerator to improve food profiling. SieveMem differs from all these works in the target kernel and can be orthogonally used in conjunction with them in any genomics pipeline that requires alignment.

VIII. Conclusion

This paper proposes a memristor-based CIM-enabled architecture for pre-alignment filters called SieveMem to (1)

accelerate shared kernels in pre-alignment filters and (2) prevent unnecessary data movement for sequence alignment by filtering dissimilar short sequences inside the main memory. The paper also discusses a CIM-friendly algorithm for pre-alignment filtering called BandedKrait that is suitable for implementation on SieveMem. Considering a larger genomics pipeline, accelerated BandedKrait on SieveMem is fast enough to shift the processing bottleneck back (again) to the DP step of the remaining sequences. Hence, our work demands even more accurate pre-alignment filtering and/or better DP-based alignment algorithms.

REFERENCES

[1] J. Ahn *et al.*, "A scalable processing-in-memory accelerator for parallel graph processing," in *ISCA*, 2015.

[2] M. Alser *et al.*, "Shouji: A Fast and Efficient Pre-Alignment Filter for Sequence Alignment," *Bioinformatics*, 2019.

[3] M. Alser *et al.*, "From molecules to genomic variations: Accelerating genome analysis via intelligent algorithms and architectures," *Computational and Structural Biotechnology Journal*, 2022.

[4] M. Alser *et al.*, "MAGNET: Understanding and Improving the Accuracy of Genome Pre-Alignment Filtering," *arXiv*, 2017.

[5] M. Alser *et al.*, "SneakySnake: A Fast and Accurate Universal Genome Pre-Alignment Filter for CPUs, GPUs, and FPGAs," *Bioinformatics*, 2020.

[6] A. Ankit *et al.*, "PUMA: A programmable ultra-efficient memristor-based accelerator for machine learning inference," in *ASPLOS*, 2019.

[7] S. S. Banerjee *et al.*, "ASAP: Accelerated short-read alignment on programmable hardware," *IEEE Transactions on Computers*, 2018.

[8] D. R. Bentley *et al.*, "Accurate whole human genome sequencing using reversible terminator chemistry," *nature*, 2008.

[9] D. S. Cali *et al.*, "GenASM: A high-performance, low-power approximate string matching acceleration framework for genome sequence analysis," in *MICRO*, 2020.

[10] P. Chi *et al.*, "PRIME: A novel processing-in-memory architecture for neural network computation in reram-based main memory," *ISCA*, 2016.

[11] M. M. Clark *et al.*, "Diagnosis of genetic diseases in seriously ill children by rapid whole-genome sequencing and automated phenotyping and interpretation," *Science translational medicine*, 2019.

[12] M. Crochemore *et al.*, "A subquadratic sequence alignment algorithm for unrestricted scoring matrices," *SIAM journal on computing*, 2003.

[13] J. Daily, "Parasail: SIMD C Library for Global, Semi-global, and Local Pairwise Sequence Alignments," *BMC bioinformatics*, 2016.

[14] S. R. Eddy, "What is dynamic programming?" *Nature biotechnology*, 2004.

[15] X. Fei *et al.*, "Fpgasw: Accelerating large-scale smith–waterman sequence alignment application with backtracking on fpga linear systolic array," *Interdisciplinary Sciences: Computational Life Sciences*, 2018.

[16] J. D. Ferreira *et al.*, "pLUTo: Enabling massively parallel computation in dram via lookup tables," in *MICRO*, 2022.

[17] C. Firtina *et al.*, "BLEND: a fast, memory-efficient and accurate mechanism to find fuzzy seed matches in genome analysis," *NAR Genomics and Bioinformatics*, 2023.

[18] C. Firtina *et al.*, "ApHMM: Accelerating profile hidden markov models for fast and energy-efficient genome analysis," *arXiv*, 2022.

[19] F. Gao *et al.*, "Computedram: In-memory compute using off-the-shelf drams," in *MICRO*, 2019.

[20] E. Georganas *et al.*, "meraligner: A fully parallel sequence aligner," in *IPDPS*, 2015.

[21] G. S. Ginsburg *et al.*, "Precision medicine: from science to value," *Health Affairs*, 2018.

[22] S. Hamdioui *et al.*, "Memristor based computation-in-memory architecture for data-intensive applications," in *DATE*, 2015.

[23] J. S. Kim *et al.*, "GRIM-Filter: Fast seed location filtering in DNA read mapping using processing-in-memory technologies," *BMC genomics*, 2018.

[24] W. Kim *et al.*, "Multistate memristive tantalum oxide devices for ternary arithmetic," *Scientific reports*, 2016.

[25] T. Lassmann *et al.*, "Kalign–an accurate and fast multiple sequence alignment algorithm," *BMC bioinformatics*, 2005.

[26] B. C. Lee *et al.*, "Phase change memory architecture and the quest for scalability," *Communications of the ACM*, 2010.

[27] S. Li *et al.*, "Drisa: A dram-based reconfigurable in-situ accelerator," in *MICRO*, 2017.

[28] S. Li *et al.*, "Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories," in *DAC*, 2016.

[29] Y. Liu *et al.*, "CUDASW++ 3.0: Accelerating Smith-Waterman Protein Database Search by Coupling CPU and GPU SIMD Instructions," *BMC bioinformatics*, 2013.

[30] Q. Lou *et al.*, "Helix: Algorithm/Architecture Co-design for Accelerating Nanopore Genome Base-calling," in *PACT*, 2020.

[31] R. Luo *et al.*, "SOAP3-dp: Fast, Accurate and Sensitive GPU-based Short Read Aligner," *PloS one*, 2013.

[32] M. L. Metzker, "Sequencing technologies—the next generation," *Nature reviews genetics*, 2010.

[33] MNEMOSENE partners, "The MNEMOSENE project." http://www.mnemosene.eu/, accessed: 2022-06-02. 2020.

[34] S. B. Needleman *et al.*, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," *Journal of molecular biology*, 1970.

[35] M. Pages-Gallego *et al.*, "Comprehensive and standardized benchmarking of deep learning architectures for basecalling nanopore sequencing data," *bioRxiv*, 2022.

[36] J. Quick *et al.*, "Real-time, portable genome sequencing for ebola surveillance," *Nature*, 2016.

[37] V. Seshadri *et al.*, "Ambit: In-memory accelerator for bulk bitwise operations using commodity DRAM technology," in *MICRO*, 2017.

[38] A. Shafiee *et al.*, "ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," *ISCA*, 2016.

[39] T. Shahroodi *et al.*, "RattlesnakeJake: A Fast and Accurate Pre-Alignment Filter Suitable for Computation-in-Memory," in *Embedded Computer Systems: Architectures, Modeling, and Simulation: 23nd International Conference, SAMOS 2023, Samos, Greece, July 2–6, 2023, Proceedings*. Springer, 2022.

[40] T. Shahroodi *et al.*, "Demeter: A fast and energy-efficient food profiler using hyperdimensional computing in memory," *IEEE Access*, 2022.

[41] T. Shahroodi *et al.*, "KrakenOnMem: a memristor-augmented HW/SW framework for taxonomic profiling," in *ICS*, 2022.

[42] J. Shendure *et al.*, "Next-generation dna sequencing," *Nature biotechnology*, 2008.

[43] A. Singh *et al.*, "Referencing-in-array scheme for rram-based cim architecture," in *DATE*, 2022.

[44] A. Singh *et al.*, "Cim-based robust logic accelerator using 28 nm stt-mram characterization chip tape-out," in *AICAS*, 2022.

[45] T. F. Smith *et al.*, "Identification of common molecular subsequences," *Journal of molecular biology*, 1981.

[46] M. Šošić *et al.*, "Edlib: A C/C++ Library for Fast, Exact Sequence Alignment Using Edit Distance," *Bioinformatics*, 2017.

[47] Synopsys, Inc., "Synopsys Design Compiler," https://www.synopsys.com/support/training/rtl-synthesis/design-compiler-rtl-synthesis.html.

[48] Y. Turakhia *et al.*, "Darwin: A genomics co-processor provides up to 15,000 x acceleration on long read assembly," *ASPLOS*, 2018.

[49] Z. Ullah *et al.*, "Z-tcam: an sram-based architecture for tcam," *TVLSI*, 2014.

[50] K. Wang *et al.*, "Low-power non-volatile spintronic memory: STT-RAM and beyond," *Journal of Physics D: Applied Physics*, 2013.

[51] R. Waser *et al.*, "Redox-based resistive switching memories–nanoionic mechanisms, prospects, and challenges," *Advanced materials*, 2009.

[52] Q. Xia *et al.*, "Memristive crossbar arrays for brain-inspired computing," *Nature materials*, 2019.

[53] L. Xie *et al.*, "Scouting logic: A novel memristor-based logic design for resistive computing," in *ISVLSI*, 2017.

[54] H. Xin *et al.*, "Shifted hamming distance: a fast and accurate simd-friendly filter to accelerate alignment verification in read mapping," *Bioinformatics*, 2015.

[55] S. Yin *et al.*, "Xnor-sram: In-memory computing sram macro for binary/ternary deep neural networks," *ISSC*, 2020.

[56] M. Zahedi *et al.*, "SparseMEM: Energy-efficient Design for In-memory Sparse-based Graph Processing," in *DATE*, 2023.

[57] M. Zahedi *et al.*, "System Design for Computation-in-Memory: From Primitive to Complex Functions," in *VLSI-SoC*, 2022.

[58] M. Zahedi *et al.*, "Efficient Signed Arithmetic Multiplication on Memristor-Based Crossbar," *IEEE Access*, 2023.