RAPIDx: High-performance ReRAM Processing in-Memory Accelerator for Sequence Alignment

Weihong Xu, Saransh Gupta, Niema Moshiri, and Tajana Rosing, *Fellow, IEEE*

***Abstract*—Genome sequence alignment is the core of many biological applications. The advancement of sequencing technologies produces a tremendous amount of data, making sequence alignment a critical bottleneck in bioinformatics analysis. The existing hardware accelerators for alignment suffer from limited on-chip memory, costly data movement, and poorly optimized alignment algorithms. They cannot afford to concurrently process the massive amount of data generated by sequencing machines. In this paper, we propose a ReRAM-based accelerator, RAPIDx, using processing in-memory (PIM) for sequence alignment. RAPIDx achieves superior efficiency and performance via software-hardware co-design. First, we propose an adaptive banded parallelism alignment algorithm suitable for PIM architecture. Compared to the original dynamic programming-based alignment, the proposed algorithm significantly reduces the required complexity, data bit width, and memory footprint at the cost of negligible accuracy degradation. Then we propose the efficient PIM architecture that implements the proposed algorithm. The data flow in RAPIDx achieves four-level parallelism and we design an in-situ alignment computation flow in ReRAM, delivering 5.5-$9.7\times$ efficiency and throughput improvements compared to our previous PIM design, RAPID. The proposed RAPIDx is reconfigurable to serve as a co-processor integrated into the existing genome analysis pipeline to boost sequence alignment or edit distance calculation. On short-read alignment, RAPIDx delivers $131.1\times$ and $46.8\times$ throughput improvements over state-of-the-art CPU and GPU libraries, respectively. As compared to ASIC accelerators for long-read alignment, the performance of RAPIDx is 1.8-$2.9\times$ higher.**

***Index Terms*—Processing in-memory, genome analysis, sequence alignment, non-volatile memory, dataflow optimization**

I. Introduction

Genome techniques are becoming increasingly crucial in various fields. Modern genome analysis techniques have been applied to human DNA to diagnose genetic diseases by identifying disease-associated structural variants [1]. The genome sequence information is also used to infer the evolutionary history of an organism over time [2]. These sequences can also be analyzed to provide information on populations of viruses within individuals, allowing for a comprehensive understanding of underlying viral selection pressures [3].

DNA sequence alignment is a key step in genome analysis that gains increasing significance due to the following reasons. First, several types of sequencing errors occur when the sequencing machine reads the genome. Additionally, genetic mutations and variations also introduce sequence differences. DNA alignment algorithms, like Needleman–Wunsch (NW) [4] and Smith-Waterman (SW) [5], are used to identify the optimal match between the query and reference sequences. The other reason is that the alignment step has become the bottleneck of genome analysis pipeline because sequence alignment is a computation-intensive and memory-intensive workload, taking up 60-80% runtime of popular genome analysis tools [6]–[9]. Therefore, boosting DNA sequence alignment plays an important role in accelerating genome analysis.

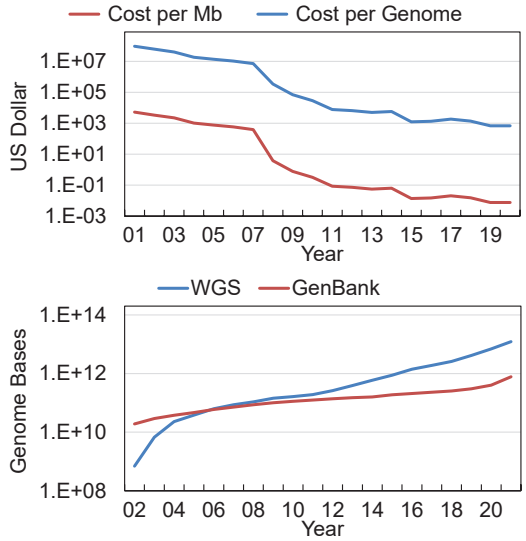


Figure 1: Trend of unit sequencing cost [16] and genome data volume [17] over the past decade.

Various algorithm optimizations have been developed for software libraries [6], [7], [9], [10]. However, the limited computing resources of CPU severely restrict the achievable performance. These works fail to generate satisfactory processing throughput and energy efficiency. To this end, many efforts have been made to design acceleration solutions on ASIC [11]–[13], GPU [8], [14], or FPGA [15] platforms. Through optimizing algorithm and hardware architecture, these accelerators have shown significant improvements in terms of efficiency and processing speed. However, the memory-intensive nature of DNA alignment algorithms makes them suffer from the limited on-chip as well as expensive data movement between off-chip memory and processing cores, incurring energy overhead caused by data movement.

The advent of high-throughput next-generation sequencing (NGS) technique [18] enlarges the gap between the processing capabilities of existing alignment accelerators and the rapidly generated genome data. Fig. 1 shows the unit cost of genome sequencing has plunged by over $10^4\times$ during the last decade. Meanwhile, the genome data volume of whole genome sequencing (WGS) and GenBank [17] have also expanded by $10^2\times$ to $10^4\times$. The genome data growth has significantly surpassed Moore's Law, meaning that acceleration solutions with orders of magnitude higher efficiency are needed for sequence alignment.

W. Xu, N. Moshiri, and T. Rosing are with University of California, San Diego (UCSD), La Jolla, CA 92093 USA (e-mail: wexu@ucsd.edu; niema@ucsd.edu; tajana@eng.ucsd.edu).

Saransh Gupta is now with IBM Research (e-mail: saransh@ibm.com).

Part of this work was presented in International Symposium on Low Power Electronics and Design (ISLPED), 2019.

Processing in-memory (PIM) is promising to mitigate the data movement issue and provides massive parallelism. This is because PIM enables in-situ data computation inside memory, thereby throttling the latency and energy of data movement [19]–[22]. Existing PIM-based accelerators for genome analysis [23]–[28] take PIM's advantages of high data parallelism and low-cost data movement, showing orders of magnitude efficiency and performance improvements over CPU and GPU.

We previously presented the PIM architecture for sequence alignment, called RAPID [28], which computes DNA alignment in memory. However, RAPID has the following deficits. First, the original DP algorithm [4] used by RAPID is sub-optimal since it is unable to measure the affine gap penalty, which has been widely used in software libraries [7], [8] and shown optimal alignment quality [29]. Second, RAPID does not consider software-hardware co-optimization, thereby wasting a large amount of energy and computing resources on redundant computations. Recent works [9], [30] demonstrate DP alignment algorithm exhibits great redundancy, and most of computation can be skipped using banded alignment [31] to accelerate the alignment process at the cost of negligible accuracy degradation. In this paper, we propose a software-hardware co-design, RAPIDx, that exploits digital PIM techniques on ReRAM to enable a highly parallel and more energy-efficient acceleration for sequence alignment. The key contributions of this work can be summarized as follows:

- **PIM-friendly dynamic programming (DP) alignment**: We consider the affine gap penalty to construct more accurate scoring functions. Then we propose the adaptive banded parallelized DP alignment that is friendly for PIM implementation. The proposed alignment algorithm reduces the required arithmetic precision from 32-bit to only 5-bit and obtains higher data parallelism. Meanwhile, the adaptive wavefront direction and bandwidth schemes significantly reduce memory footprint and computational complexity by over $10\times$ at the cost of $< 0.15\%$ accuracy loss.
- **High-performance PIM architecture**: We propose efficient PIM architecture for RAPIDx, which achieves four-level data parallelism. RAPIDx leverages in-situ PIM operations [32] to perform low-energy and row-parallel in-memory alignment. Our peripheral circuits implement fast traceback as well as complex functions not friendly for PIM. Compared to previous RAPID [28], RAPIDx shows $5.5\times$ latency reduction and $6.2\times$ energy improvements.
- **System optimization and reconfigurable design**: We design novel PIM computing operations that are reconfigurable to support multiple types of alignment scoring as well as edit distance computation. This makes RAPIDx a multi-purpose accelerator that is flexible to support alignment and edit distance computations. We also analyze several possible limiting factors when integrating RAPIDx into existing computing system, including ReRAM cell's limited endurance, switching speed, and system considerations.
- **Improvements and accelerations**: We compare RAPIDx with state-of-the-art CPU baselines (Minimap2 [7] and Edlib [6]), GPU baseline (GASAL2 [8]), and ASIC baselines (ABSW [11] and GenASM [12]) on various workloads. For short-read alignment, RAPIDx delivers an average $131.1\times$ and $46.8\times$ higher throughput compared to Minimap2 [7] and GASAL2 [8], respectively. For long-read alignment, $1.8\times$ to $2.9\times$ throughput improvements are observed over ABSW [11] and GenASM [12]. For edit distance calculation, RAPIDx obtains up to $321\times$ speedup over Edlib [6].

II. Related Work

A. Software for Sequence Alignment

Several software libraries [6]–[9] have been developed for boosted genome analysis. The main point is optimizing the SW algorithm and CPU/GPU datapath to deliver accurate and fast sequence alignment. BWA-MEM [9] is software to map DNA sequences against large reference genomes. BWA-MEM aligns the given sequences using Burrows-Wheeler Transform (BWT) [33]. However, the memory footprint of aligning long genome is large and the irregular memory access of BWT limits the processing speed. Edlib [6] is a C++ library that exploits Myers's bit-vector algorithm [34] to parallelize the SW-based alignment. To realize more accurate and efficient alignment, Minimap2 [7] introduces two promising optimization strategies, banded alignment [31] and difference-based SW [35], which can be fitted into the datapath of single instruction, multiple data (SIMD). Minimap2 generates over $10\times$ speedup over BWA-MEM. Even though these software libraries achieve fine-grain optimization, the limited computing resources on CPU fail to provide opportunities for further acceleration. Some researchers shift the focus to GPU-based acceleration. CUDAlign 4.0 [14] increases the parallelism by splitting each SW alignment into multiple GPUs and reducing the data dependency of the traceback process. GASAL2 [8] optimizes the data organization and develops efficient kernels for multiple sequence alignment workloads. These libraries exploit the abundant computing resources on GPU. But the resulted efficiency is not high because optimizations for SW algorithms are lacked due to the architectural limitations of GPU. In this work, RAPIDx is a software and hardware co-design that realizes algorithm and hardware optimizations at the same time.

B. Hardware Acceleration for Sequence Alignment

ASIC Accelerator: Various hardware accelerators [11]–[13], [23]–[25], [27], [32] have be presented to obtain higher energy efficiency and speedup for genome analysis. For ASIC designs, one challenge is how to realize long-read alignment under the constraints of limited on-chip memory. Darwin [13] proposes near-optimal tiling methods to align arbitrary sequence lengths, only requiring constant memory space. ABSW [11] leverages the tiling schemes [13] and implements an adaptively banded alignment on ASIC, achieving significant efficiency improvement. GenASM [12] proposes an approximate string matching algorithm and a systolic-array-based accelerator to increase data parallelism while reducing memory footprint. Although prior works employ a variety of optimizations, the limited on-chip memory is still the bottleneck when aligning long sequences.

PIM Accelerator: PIM is a promising solution to increase data parallelism and energy efficiency via computing data in situ [21], [26], [27], [36]. The PIM-based alignment designs proposed in PRINS [23] and BioSEAL [24] accelerate algorithms using resistive content addressable memory (CAM).

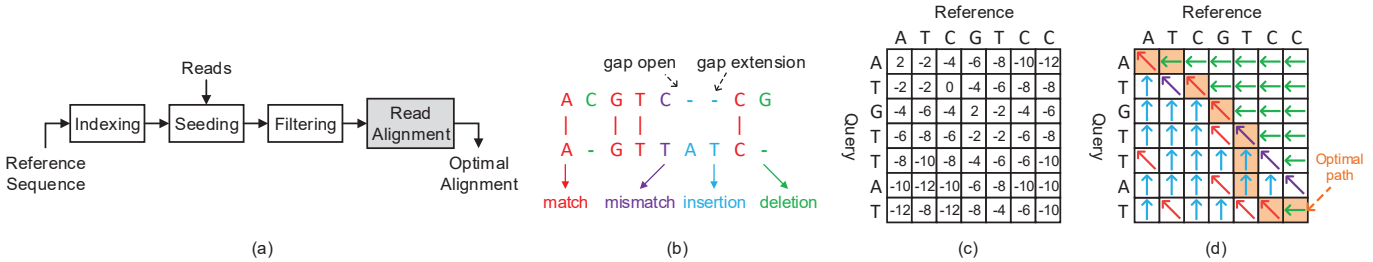


Figure 2: (a) The pipeline of genome sequence analysis. (b) Alignment example of sequences ACGTCCG and AGTTATC with affine gap penalties, (c) Score matrix, (d) Traceback matrix.

But the sequential associative search incurs a large amount of write operation and internal data movement, degrading efficiency, lifetime, and storage efficiency. Another set of works accelerates short read alignment, where long sequences are broken down into smaller sequences and heuristic methods are applied. AlignS [26], AligneR [27] and PIM-Aligner [36] exploit FM-index algorithm and PIM to realize short-read alignment. However, FM-index incurs irregular memory access, and is hard to exploit the data parallelism of PIM. RAPID [28] is a ReRAM-based PIM accelerator to implement in-situ alignment computation in the memory, which drastically reduces the data movement. However, the adopted algorithm in RAPID is sub-optimal and requires quadratic complexity, limiting its capability of aligning long sequences. In this work, we present several optimizations for alignment algorithms and hardware architecture to fully leverage the highly parallel PIM while providing satisfactory alignment quality. Our design, RAPIDx, delivers up to $9.3\times$ alignment efficiency improvement compared to other PIM baselines.

III. Background

A. Genome Sequence Analysis

1) Overall Pipeline

A typical pipeline of modern genome sequencing analysis [7], [9], [10] involves indexing, seeding, filtering, and read alignment steps as shown in Fig. 2 (a). For the indexing phase, the entire reference sequence is stored into special data structures, like BWT [33] and FM-indexing. The indexing is for quickly obtaining the location of query sequence in the reference sequence. Then, the seeding process uses the indexing information to query the potential mapping locations of genome reads. The filtering step discards invalid candidates or combines nearby candidates from the seeding step. Finally, the genome reads are aligned against the reference sequence around the candidate location using the SW algorithm. Among these steps, the most time-consuming step is read alignment used to determine how the read sequence can be optimally mapped to the reference sequence.

2) Sequence Alignment with Affine Gap Penalty

The sequence alignment can be described as finding the maximum alignment score between the reference sequence $R = r_1, r_2, ..., r_m$ and the query sequence $Q = q_1, q_2, ..., q_n$. Natural evolution and mutation as well as experimental errors during sequencing poses two types of changes in sequences - substitutions and indels. A substitution changes a base of the sequence with another, leading to a mismatch whereas an indel either inserts or deletes a base. Fig. 2 (b) shows the comparison of two sequences, $R = \mathsf{ACGTCCG}$ and $Q = \mathsf{AGTTATC}$. The left part rigidly compares the ith base of Q with R, where match and mismatch are considered. The right part assumes a different alignment that involves insertion and deletion. Note that the notation of dashes $(-)$ is conceptual, and are used to illustrate a potential scenario that one sequence has been (or can be) evolved to the other.

Most sequence alignments are categorized into global or local alignment. The global and local alignments can be optimally addressed by NW algorithm [4] and SW algorithm [5], respectively. NW and SW both build up and compute the optimal alignment sequence based on DP [37], [38]. DP-based methods involve forming alignment matrices, which are used to compute scores of various alignments based on a pre-defined scoring function. The scoring function is essential for accurate alignment since it is used to update the scoring matrix in DP. The previous work [39] mostly uses the scoring function with linear gap penalty, where the penalty is increasing linearly with the gap length. However, the linear gap penalty is insufficient to accurately evaluate the alignment scores for those sequences with the same total gap length. The gap-less sequence is normally more biologically meaningful compared to the sequence with more gaps. In this work, we adopt the scoring function with affine gap penalties [40] that consider the number and length of gaps. Fig. 2 shows an example of alignment between sequence $R = \mathsf{ACGTCCG}$ and $Q = \mathsf{AGTTATC}$ using affine gap penalties. The updating rules for scoring matrices in DP with affine gap penalty can be expressed as:

$$E_{i,j} = \max \begin{cases} H_{i-1,j} - o \\ E_{i-1,j} - e \end{cases} \quad F_{i,j} = \max \begin{cases} H_{i,j-1} - o \\ F_{i,j-1} - e \end{cases} \quad (1)$$
$$H_{i,j} = \max\{E_{i,j}, F_{i,j}, H_{i-1,j-1} - s(r_j, q_i)\}$$

where E and F denote the alignment matrices that store the scores of insertion and deletion, respectively. H is the alignment score matrix that stores the total scores. $s(r_j, q_i)$ denotes the score of match A or mismatch B by comparing r_j and q_i. The gap opening penalty is o while e denotes the gap extension penalty. Fig. 2 (c) shows an example of score matrix H calculated using Eq (1) with penalties $A = 2, B = 4, o = 4, e = 2$. A traceback phase is required to construct the optimal alignment path after the computation for all alignment matrices. The traceback matrix in Fig. 2 (d) stores the path information. For global alignment, the traceback starts from the cell at the bottom-right corner while local alignment starts from the cell with the maximum score.

B. Difference-based Dynamic Programming (DP) Alignment

The updating function in Eq. (1) has the following limitations. The maximum value in the alignment matrix scales up linearly with the matrix dimension. The data bit width needs to be increased as the sequence length increases to avoid computation overflow. Previous accelerations [11], [13] use a fixed bit width in the worst case, resulting in low computation efficiency. To resolve this issue, the original DP updating is rewritten into a computation-efficient form, named the difference-based formulation [35]. The basic idea is to store and compute the value difference of adjacent elements instead of the full-precision value in the alignment matrix, thus reducing the required arithmetic precision. As shown in the left side of Eq. (2), four matrices ΔH, ΔV, ΔE, and ΔF are used to store the difference values. After substituting the four difference matrices into Eq. (1), the alignment matrices (H, E, and F) are converted into the following difference-based formulation:

$$\begin{cases} \Delta H_{i,j} = H_{i,j} - H_{i-1,j} \\ \Delta V_{i,j} = H_{i,j} - H_{i,j-1} \\ \Delta E_{i,j} = E_{i+1,j} - H_{i,j} \\ \Delta F_{i,j} = F_{i,j+1} - H_{i,j} \end{cases}$$

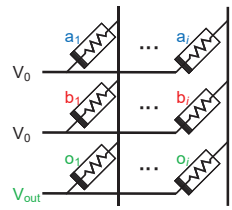
$$\Rightarrow \begin{cases} A_{i,j} = \max \begin{cases} s(i,j), \\ \Delta E_{i-1,j} + \Delta V_{i-1,j}, \\ \Delta F_{i,j-1} + \Delta H_{i,j-1} \end{cases} \\ \Delta H_{i,j} = A_{i,j} - \Delta V_{i-1,j} \\ \Delta V_{i,j} = A_{i,j} - \Delta H_{i,j-1} \\ \Delta E_{i,j} = \max\{-o, \Delta E_{i-1,j} - \Delta H_{i,j}\} - e \\ \Delta F_{i,j} = \max\{-o, \Delta F_{i,j-1} - \Delta V_{i,j}\} - e \end{cases} \quad (2)$$

where an intermediate variable $A_{i,j}$ is added to the computation. It should be noted that Eq. (2) only changes the expression of original DP in Eq. (1) while retaining the identical information. Eq. (2) can generate the identical alignment results as Eq. (1).

There are two benefits of the difference-based alignment in Eq. (2). First, the arithmetic precision requirement is significantly reduced. According to [7], [35], the data range of $\Delta H_{i,j}$ and $\Delta V_{i,j}$ are bounded by $[-o-e, -e]$ while $\Delta E_{i,j}$ and $\Delta F_{i,j}$ are bounded by $[-o-e, M+o+e]$, where M denotes the maximum value of $s(i,j)$. Compared to the full-precision alignment, the difference-based representations only needs $\lceil \log_2(M+2o+2e+1) \rceil$-bit integer to calculate the alignment. Second, the required data precision is only determined by the used affine gap scores while independent with the sequence length. This property allows us to use a unified data bit width for different sequence lengths. For example, we use 5-bit integer for computing alignment and 3-bit integer for calculating edit distance as introduced in Section V-D.

C. Digital Processing In-Memory (PIM)

Various types of memory devices are used for PIM to resolve the "memory wall" problem, including MRAM [26], [36], PCM, and SRAM [41]. MRAM suffers from severe read disturbance when the memory density increases [42]. ReRAM has higher memory density than MRAM and SRAM because the ReRAM cell is much smaller than MRAM and SRAM. Moreover, ReRAM has lower leakage power compared to other devices, making it an energy-efficient candidate for PIM. FeFET [43] and NAND flash [44] are the other two potential PIM candidates that are still in early development phase while



(a)

a_i	b_i	o_{init}	o_i
0	0	1	1
0	1	1	0
1	0	1	0
1	1	1	0

(b)

Figure 3: Implementing NOR operation using ReRAM-based digital processing in-memory (PIM).

ReRAM has been physically verified at scale [45]. ReRAM has higher error rates, but this is not a significant issue for alignment as alignment algorithms are already statistical in nature, and can tolerate significant errors at bit level. Considering all these benefits, we choose ReRAM-based PIM in this work.

Traditionally, PIM with memristors is based on reading currents through different cells. However, some recent work has demonstrated ways, both in literature [32], [46], [47] and by fabricating chips [48], to implement logic using memristor switching. Digital PIM exploits variable switching of memristors. The output device switches whenever the voltage across it exceeds a threshold [49]. This property can be exploited to implement a variety of logic functions inside memory [32], [46]. Fig. 3 shows an example of implementing NOR operation using ReRAM-based PIM [32]. A voltage V_0 is in parallel applied to the rows that contain the operand cells a_i and b_i. The output cell o_i switches to low voltage status (logical '0') from initial logical '1' whenever one or more inputs are '1's, resulting in logical `NOR` operation. Since `NOR` is a functionally complete logic gate, it can be used to implement other logic operations like addition [46] and multiplication [50]. For example, 1-bit addition (inputs being A, B, C) can be represented in the form of `NOR` as:

$$\begin{aligned} C_{out} &= ((A+B)' + (B+C)' + (C+A)')' \\ S &= (((A'+B'+C')' + ((A+B+C)' + C_{out})')')' \end{aligned} \quad (3)$$

where C_{out} and S are the generated carry and sum bits of addition. $(A+B+C)'$, $(A+B)'$, and A' represent $NOR(A,B,C)$, $NOR(A,B)$, and $NOR(A,A)$, respectively.

In-memory operations are in general slower than the corresponding CMOS-based implementations because memristor devices switch slowly. However, PIM architectures can provide significant speedup when it is exposed massive parallelism. Meanwhile, the long processing latency is amortized due to the high parallelism. In this work, RAPIDx utilizes two types of PIM operations (XOR and addition) introduced in FELIX [32] to perform in-memory alignment computation. This is because FELIX's PIM primitives achieve the same or significantly better latency, memory consumption, and efficiency than other digital PIM schemes [46], [51]. The other digital PIM scheme [52] for floating-point arithmetic is not suitable for the fixed-point arithmetic in RAPIDx.

Specifically, the XOR and 1-bit addition are realized through:

- **XOR:** XOR ($\oplus$) can be expressed by OR ($+$), AND (.), and NAND ($(.)'$) as $A \oplus B = (A+B).(A.B)'$. We first calculate OR and then use its output cell to implement NAND. This operation is executed in parallel over all the columns of two rows. This logic just requires 2 cycles and one additional

memristor device, which acts as the output cell.

- **Addition:** Let A, B, and C_{in} be 1-bit inputs of addition, and S and C_{out} the generated sum and carry bits respectively. Then, S is implemented as two serial in-memory XOR operations (A⊕B)⊕C. C_{out}, on the other hand, can be executed by inverting the output of the Min function proposed in [32]. Addition takes a total of 6 cycles and similar to XOR, we parallelize it over all columns in two rows.

IV. Efficient Alignment in RAPIDx

In this section, we first analyze the challenges of realizing efficient in-memory alignment using digital PIM. Then we propose the adaptive banded parallelized DP alignment to balance performance and accuracy loss.

A. Challenges of Alignment using PIM

1) Data Bit Width and Latency

Compared to CMOS-based circuits, the slow switching speed of ReRAM cells incurs long latency when implementing PIM operations in Section III-C. For example, 1-bit PIM addition takes 6 to 12 clock cycles [32]. As discussed in Section III-B, the data bit width and range grow linearly with the sequence length. The previous accelerators [24], [28] adopt the original DP algorithm which uses 32-bit integers to guarantee lossless alignment. However, 32-bit integer is over-designed and incurs long processing latency when aligning short sequences (<1kbp) since the lower 12-bit width is enough to provide sufficient data dynamic range [11]. Therefore, developing an alignment algorithm using low bit-width data is beneficial to reduce PIM latency. The difference-based DP alignment in Section III-B is a potential solution to alleviate this as it needs fixed data width independent of sequence length.

2) Data Parallelism

ReRAM-based PIM architectures [24], [25], [28], [36] offer substantial opportunities of extending the data parallelism. High parallelism amortizes the incurred long latency of PIM operations. One example is the row-parallel PIM operation [24], [32], where the bit-serial computation can be performed in the entire memory row simultaneously. How to exploit the architectural parallelism of ReRAM is key to attaining high alignment throughput. The other challenge from the algorithm is how to expose enough parallelism to ReRAM. For DP alignment, adjacent cells in alignment matrices exhibit data dependency. Previous works [7], [11], [13], [24], [35] utilize the wavefront parallelism based on the fact that cells over anti-diagonal have no data dependency. Unfortunately, this parallelism is far enough for PIM architecture.

3) Complexity and Accuracy

Fig. 4 (a) illustrates the full DP alignment using Eq. (1), where all cells in the matrices with shape $m \times n$ need to be computed (m and n denote the lengths of reference and query sequences, respectively). The complexity is prohibitive when aligning long sequences. Banded alignment [30], [31] is an effective method to reduce the complexity from quadratic to near-linear. It should be noted that the banded approach is an approximate algorithm that may introduce accuracy degradation. One simple solution is to use a fixed and wide bandwidth ($B = 128$) as [11]. But this degrades the throughput and performance gain since wider bandwidth leads to higher complexity. The challenge is how to select narrow bandwidth for various lengths while ensuring the optimality of results.

B. Adaptive Banded Parallelized DP Alignment

We propose the adaptive banded parallelized DP alignment to resolve the above-mentioned challenges. The difference-based alignment in Eq. (2) relaxes the requirement of data precision and reduces the bit width for DP alignment. However, the computation of $\Delta H_{i,j}, \Delta V_{i,j}, \Delta E_{i,j}$, and $\Delta F_{i,j}$ can only be accomplished in a serial manner. Specifically, $A_{i,j}$ needs to be first computed before updating $\Delta H_{i,j}$ and $\Delta V_{i,j}$. Then the values of $\Delta V_{i,j}$ and $\Delta E_{i,j}$ require the newly updated $\Delta H_{i,j}$ and $\Delta V_{i,j}$. Consequently, parallelizing the computation for each updating step is difficult due to the inherent data dependency. We resolve this issue through further transforming Eq. (2) into a parallelized version similar to [35]. The variables in Eq. (2) are rewritten as the top part of Eq. (4), where auxiliary o and e values are added to each variable in Eq. (2). After substituting it into Eq. (2), we have the parallelized difference-based alignment as follows:

$$\begin{cases} A'_{i,j} = A_{i,j} + 2o + 2e \\ \Delta H'_{i,j} = \Delta H_{i,j} + o + e \\ \Delta V'_{i,j} = \Delta V_{i,j} + o + e \\ \Delta E'_{i,j} = \Delta E_{i-1,j} + \Delta V_{i-1,j} + 2o + 2e \\ \Delta F'_{i,j} = \Delta F_{i,j-1} + \Delta H_{i,j-1} + 2o + 2e \end{cases} \\ \Rightarrow \begin{cases} A'_{i,j} = \max\{s'(i,j), \Delta E'_{i-1,j}, \Delta F'_{i,j-1}\} \\ \Delta H'_{i,j} = A'_{i,j} - \Delta V'_{i-1,j} \\ \Delta V'_{i,j} = A'_{i,j} - \Delta H'_{i,j-1} \\ \Delta E'_{i,j} = \max\{A'_{i,j}, \Delta E'_{i-1,j} + o\} - \Delta H'_{i,j-1} \\ \Delta F'_{i,j} = \max\{A'_{i,j}, \Delta F'_{i,j-1} + o\} - \Delta V'_{i-1,j} \end{cases} \tag{4}$$

where $\Delta H'_{i,j}$ and $\Delta V'_{i,j}$ only depend on new $A'_{i,j}$ and previous $\Delta V'_{i-1,j}$ and $\Delta H'_{i,j-1}$, respectively. Likewise, $\Delta E'_{i,j}$ and $\Delta F'_{i,j}$ can be calculated by the old $\Delta H'_{i,j-1}$ and $\Delta V'_{i-1,j}$ from the previous iteration. In this case, the relaxed data dependency between four alignment matrices provides higher computation parallelism. After obtaining $A'_{i,j}$, the computation of $\Delta H'_{i,j}$, $\Delta V'_{i,j}$, $\Delta E'_{i,j}$, and $\Delta F'_{i,j}$ can be conducted in parallel to shorten the processing latency. We call this the alignment matrix level parallelism. The data range of four alignment matrices is shifted to $[0, M + 2o + 2e]$ from $[-o - e, M + o + e]$, requiring the same bit width as Eq. (2).

The banded alignment [31] significantly reduces the complexity based on the observation that the optimal alignment path normally locates not far away from the diagonal of alignment matrices. The reduction is achieved by limiting the cells in alignment matrices that need to be computed. Fig. 4 (b) shows the banded DP alignment that only computes the cells located within a bandwidth $B = 6$ of the diagonal, whereas the rest cells are inactivated. In this way, only B wavefront cells (the cells that are updated simultaneously) are computed and moved over the main diagonal in each iteration. Bandwidth and wavefront direction are the two key factors that determine the accuracy and efficiency of banded alignment. The adaptive banded parallelized alignment adopted by RAPIDx is adaptive in the sense of bandwidth and wavefront direction as follows:

1) Adaptive Bandwidth

A narrow bandwidth $B \ll m, n$ helps to perform a low-complexity alignment as the banded DP has $\mathcal{O}(mB)$ com-

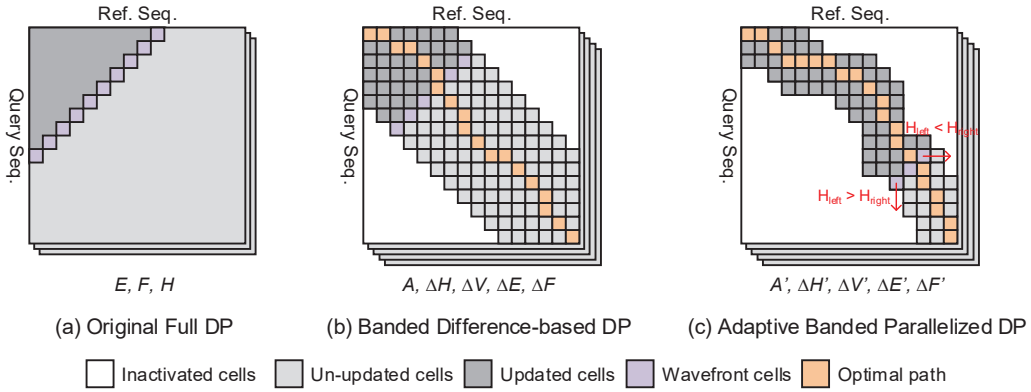


Figure 4: Illustration of three variants of DP alignment algorithms. Bandwidth $B = 6$ in (b) and $B = 3$ in (c).

plexity. To balance the algorithm efficiency and accuracy, the bandwidth B used in RAPIDx is adaptive based on the processed sequence length. The other factor to consider when choosing the bandwidth is the inflexibility of ReRAM-based PIM. The proper bandwidth needs to be determined before alignment computation. To this end, we express the relationship between bandwidth B and sequence length L as $B = \min(w + 0.01 \times L, 100)$, where w denotes the base bandwidth that determines the narrowest bandwidth while B is set to the multiple of w. The function limits the maximum bandwidth to 100 because previous BWA-MEM [9] shows $B = 100$ is enough to guarantee optimal alignment for all sequence lengths. On the other hand, a band with less than 20 is enough for over 99% cases as demonstrated in [30] but a too narrow band may not guarantee the optimality of alignment for long reads. This is because current long-read techniques (see Table II) incur much more errors and the narrow band can not fully cover the optimal path. Thus, we empirically select the 0.01 coefficient to adaptively determine the minimum bandwidth that provides negligible degradation according to L. Based on the length of the given sequences, the bandwidth B can be pre-determined before alignment. We provide detailed experiments in Section VI-B to guide the selection of the 0.01 coefficient and the best w that only introduce negligible accuracy loss.

2) Adaptive Wavefront Direction

The wavefront cells in Fig. 4 (b) and (c) can move either rightward or downward in each iteration. The alignment tools, like Minimap2 [7] and BWA-MEM [9], mostly use a pre-defined direction in Fig. 4 (b), such that the wavefront moves towards the main diagonal. When we use narrow bandwidth ($B = 3$) in Fig. 4 (c), simply computing the wavefront over the main diagonal may not obtain the optimal results because the fixed wavefront direction lacks flexibility and is unable to cover the optimal path. To this end, we use a simple adaptive wavefront direction scheme to dynamically adjust the moving direction of wavefront cells as in Fig. 4 (c). The direction is decided based on the comparison result of two edge cells in the band of score matrix. Specifically, if the value of the rightmost cell is greater than the leftmost cell, this suggests the optimal path is more likely to go rightward [53]. Hence, the current wavefront is moved rightward. Otherwise, the wavefront is moved downward. The adaptive wavefront direction scheme only needs one comparison each iteration but effectively improves the accuracy of long-read alignment according to our test results in Table V.

Table I: Comparison of DP alignment algorithms in Fig. 4

Algorithm	Complexity		Critical Path	Accuracy
	Computation	Memory		
Full DP	$\mathcal{O}(mn)$	$\mathcal{O}(mn)$	5×32 bit	High
Banded Difference-based DP	$\mathcal{O}(mB)$	$\mathcal{O}(mB)$	8×5 bit	Low
Adaptive Banded Parallelized DP	$\mathcal{O}(mB)$	$\mathcal{O}(mB)$	4×5 bit	High

We conduct an algorithmic analysis for the aforementioned DP algorithms and compare their complexity, data parallelism, and critical path in Table I. The critical path is defined as the longest data path needed to accomplish one iteration of cell updating. Thanks to the alignment matrix parallelism, the proposed adaptive banded parallelized alignment only needs half of the critical path of Eq. (2). More importantly, the adaptive wavefront direction compensates for the accuracy loss caused by narrow bandwidth, allowing the proposed algorithm to generate near-optimal results using near-linear complexity.

V. In-Memory Architecture of RAPIDx

We propose the PIM-based ReRAM accelerator, RAPIDx to implement the adaptive banded parallelized DP alignment in Section IV. RAPIDx utilizes the in-site PIM-based alignment algorithm and the efficient data flow with four-level parallelism to boost alignment process.

A. Overview

As shown in ❶ of Fig. 5, RAPIDx is a ReRAM-based PIM accelerator for genome sequence alignment. The algorithm in Section IV-B exhibits various data parallelisms, including wavefront and alignment matrix levels. RAPIDx is organized in a multi-level hierarchy to extend the data parallelism. RAPIDx consists of 64 tiles, each RAPIDx tile independently receiving and transferring genome data through global I/O buffer and global row driver. The read genome sequences are stored in the sequence buffer within each tile. To minimize the data movement, the forward DP cells updating and traceback computation happen locally in each tile. There is no communication between tiles. We conduct design space exploration in Section VI-C to choose the hardware configurations resulting in the best efficiency.

Fig. 5-❷ shows the internal structure of RAPIDx tile, where one computation memory (CMs) and multiple traceback memories (TBMs) are implemented. One CM is connected to 15 TBMs through the H-tree connection, allowing low-latency

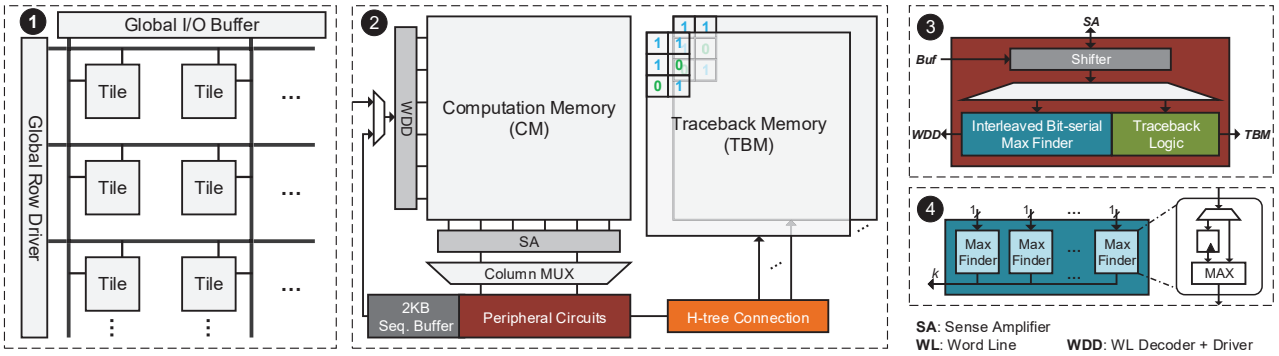


Figure 5: RAPIDx architecture. ❶ ReRAM memory organization of RAPIDx. ❷ Internal architecture of RAPIDx tile. ❸ Peripheral circuits (shifter, interleaved bit-serial max finder, and traceback logic). ❹ Interleaved bit-serial max finder.

and high-bandwidth data transfer between CMs and TBMs. The number of TBM is more than CM because most of the memory is used for storing traceback information. Each CM fetches the reference and query sequences from the 2KB sequence buffer. Then CM calculates A', $\Delta H'$, $\Delta V'$, $\Delta E'$, and $\Delta F'$ matrices in Eq. (4) using PIM-based XOR and addition operations combined with peripheral circuits. Each CM is able to access TBMs and transfer traceback data through the H-tree routing. Although the ReRAM subarray exhibits high data parallelism, some computations of alignment and traceback can not be efficiently realized in CM. For example, finding the point-wise maximum values of two vectors in [28] is complex, requiring both leading one detector and bit-wise logical operations. PIM operations [32] is unable to support low-latency traceback in Eq. (5) as well as the adaptive wavefront direction scheme. In RAPIDx, we connect peripheral circuits to sense amplifier (SA) and offload these operations to the peripheral circuits, consisting of the shifter, interleaved bit-serial max finder, and traceback logic as shown in Fig. 5-❸ and ❹.

In the peripheral circuits, we identify the max finder accounts for the largest area and has the most complex structure. The design of max finder faces several challenges. First, the additional overhead should be as low as possible to ensure not significantly sacrificing ReRAM memory density. Second, the max finder should match the processing rate of CM while minimally impacting the overall throughput. The max finding scheme in [28] incurs long latency. We further reduce the latency by offloading the max finding to the interleaved bit-serial max finder in Fig. 5-❹. The interleaved bit-serial max finder is composed of k bit-serial max finders and the width k equals to the SA's bit width. This is to match the data rate of SA. The classic bit-serial max finder receives 2-bit input in parallel. However, only 1-bit data of multiple data points in the same vector can be read from CM through SA due to CM's bit-serial data organization. Hence, we add a latch and MUX before the input of bit-serial MAX finder to make it support the comparison of bit-serial data.

B. Data Flow with Four-level Data Parallelism

To fully exploit the acceleration opportunities and increase throughput, RAPIDx achieves four-level parallelism, namely tile level, sequence level, wavefront level, and alignment matrix level, as illustrated in Fig. 6. On the host side, query reads are seeded and filtered in a batched processing manner. Then the resulted kt batches of reference and query pairs are sent to RAPIDx, where k denotes the number of memory segments in Fig. 6 (b) and t denotes the number of tiles. The kt batches of reference and query data are evenly distributed to each tile. The tile-level parallelism enables different RAPIDx tiles to process and align k independent sequences in parallel, allowing the performance of RAPIDx to scale almost linearly with the number of implemented tiles. The CM subarray with size 1024×1024 used in this paper introduces long latency due to the slow PIM operations [32]. The genome sequences in each CM are processed in batch to amortize the long latency of PIM. As illustrated in Fig. 6 (b), each CM processes a reference and a query batch with batch size k. The CM is horizontally divided into k memory segments to compute the k pairs of reference and query sequences in parallel. The column width of each memory segment equals the bandwidth B of banded alignment. Hence, there are at most $\lfloor \frac{1024}{B} \rfloor$ memory segments.

RAPIDx achieves wavefront-level and alignment matrix-level parallelism in the memory segments of CM. The wavefront-level parallelism is based on the fact that the cells over anti-diagonal have no data dependency since they only depend on the cells in the previous diagonal. The row-parallel operations of ReRAM subarray compute and update the B wavefront cells over the anti-diagonal in Fig. 4 (c) simultaneously. Meanwhile, the relaxed data dependency of parallelized alignment in Eq. (4) provides the alignment matrix-level parallelism, where $\Delta H'_{i,j}$, $\Delta V'_{i,j}$, $\Delta E'_{i,j}$, and $\Delta F'_{i,j}$ can be computed in parallel.

C. In-memory Alignment

1) Forward DP Updating

As shown in Fig. 6 (c), the data in ReRAM subarray are organized in the bit-serial manner, where each b-bit data lies vertically in b consecutive rows over the bit line. The rows of each memory segment are vertically divided into two regions, including sequence rows and processing rows. The sequence rows are used for storing DNA bases of reference and query. Before starting the wavefront cells updating, the DNA bases related to B wavefront cells are fetched from the sequence buffer and written to the sequence rows. Since each DNA base, A, G, C, T, is encoded with 2-bit data, the sequence rows occupy 4 memory rows. The rest of memory rows work as processing rows and reserved rows, which are responsible for updating wavefront cells of $A'_{i,j}, \Delta H'_{i,j}, \Delta V'_{i,j}, \Delta E'_{i,j}, \Delta F'_{i,j}$ in Eq. (4) using PIM operations [32]. The processing rows are partitioned into five

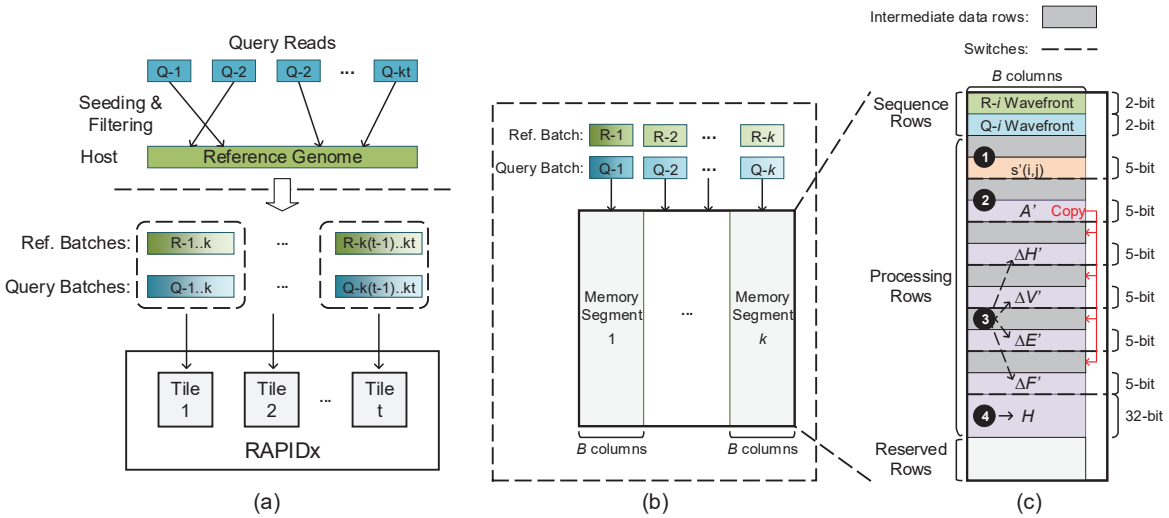


Figure 6: Four-level data parallelism and in-memory alignment in RAPIDx: (a) Tile-level parallelism. (b) Batched alignment in CM using sequence-level parallelism, (c) PIM-based in-situ banded parallelized alignment in each memory segment of CM.

partitions by switches and $A'_{i,j}, \Delta H'_{i,j}, \Delta V'_{i,j}, \Delta E'_{i,j}, \Delta F'_{i,j}$ are stored and processed in each partition. Intermediate data rows are inserted into the processing rows to store constant values and intermediate results during computation. The constants used for comparison and subtraction when updating the DP alignment include $2o + 2e$ and o. These pre-defined values are replicated and pre-stored in the reserved rows. PIM operations can directly access these values whenever needed. Specifically, the forward DP updating is computed in the following orders:

1) First, the 5-bit data $s'(i,j)$ are computed by comparing reference and query wavefront sequences (see ❶ of Fig. 6 (c)). $s'(i,j)$ requires one comparison and addition to generate the match or mismatch score. The comparison between genome bases is done using 2-bit XOR operations.
2) Second, $A'_{i,j}$ is obtained from the maximum value of $s'(i,j)$, $\Delta E'_{i-1,j}$, and $\Delta F_{i,j-1}$ as shown in ❷ of Fig. 6 (c). Two $\max$ operations are needed in this step.
3) Third, four copies of $A'_{i,j}$ are written to the intermediate data rows related to $\Delta H'_{i,j}, \Delta V'_{i,j}, \Delta E'_{i,j}$, and $\Delta F'_{i,j}$ as ❸ of Fig. 6 (c).
4) Third, $\Delta H'_{i,j}$ and $\Delta V'_{i,j}$ are updated in parallel using copied $A_{i,j}$ and previous $\Delta V'_{i-1,j}, \Delta H'_{i,j-1}$. Meanwhile, $\Delta E'_{i,j}$ and $\Delta F'_{i,j}$ are updated in parallel based on copied $A'_{i,j}$ and $\Delta E'_{i-1,j}$, $\Delta F'_{i,j-1}$, $\Delta H'_{i,j-1}$, $\Delta V'_{i-1,j}$ of the previous iteration. This step needs four subtractions, two additions, and two $\max$ operations.
5) Finally, the alignment score matrix $H_{i,j}$ need to be retrieved using the function $H_{i,j} = H_{i-1,j} + \Delta H_{i,j} = \Delta H'_{i,j} - (o + e) + H_{i-1,j}$, which requires one 5-bit subtraction and one 32-bit addition.

2) Adaptive Wavefront Direction

After wavefront cells are computed, the band will move either downwards or rightwards by one cell. Fig. 7 illustrates how the wavefront with bandwidth $B = 3$ is moved using peripheral circuits, where the wavefront direction is controlled by the shifter and sequence buffer. The max finder first compares the leftmost and rightmost cells in score matrix H, determining the next direction for wavefront. Then, the shifter receives the

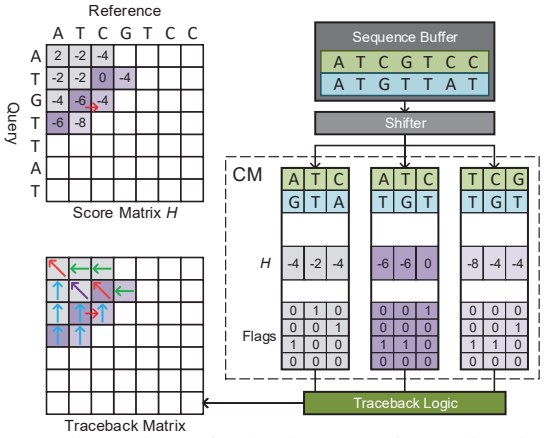


Figure 7: Illustration of adaptive wavefront direction and traceback process using peripheral circuits.

direction signal and reads the corresponding genome sequence from the sequence buffer. If the wavefront is moving rightwards, the shifter fetches reference data. Otherwise, it fetches query data. After shifting to the position of wavefront cells, the new genome sequence is written to the sequence rows within CM. In this way, the majority of computation data stay stationary in CM using in-situ PIM-based alignment, reducing the data movement overhead.

3) Traceback Process

Each iteration of DP alignment is followed by updating traceback matrix. Eq. (1) can easily compute the traceback matrix through comparing the corresponding values of three alignment matrices I, D, and H. However, the difference-based DP alignment in Eq. (2) and Eq. (4) only store the difference values and do not explicitly give the score matrix H. Therefore, we modify the formula of generating traceback information of the original DP to calculate the traceback matrix TB as the following equation:

$$TB_{i-1,j-1} = \begin{cases} 00, \text{if } s'_{i,j} == (A + o + e) \text{ or } (-B + o + e) \\ 01, \text{if } \Delta H'_{i,j} == \Delta E'_{i-1,j} - \Delta V'_{i-1,j} \\ 10, \text{if } \Delta H'_{i,j} == \Delta F'_{i,j-1} - \Delta H'_{i,j-1} \\ 11, \text{if others} \end{cases} \quad (5)$$

where two subtractions and four comparisons are needed. 00, 01, and 10 denote the cases of match or mismatch, deletion, and insertion, respectively.

As shown in Fig. 7, to efficiently implement Eq. (5) in memory, the traceback logic in ❹ of Fig. 5 reads out the 4-bit flags that indicate the traceback information from CM in a bit-serial order. Then the traceback logic converts the 4-bit flags into 2-bit traceback data and stores them into TBM. Since there will be only one "1" in the 4-bit flags. The conversion from 4-bit flags to 2-bit data is accomplished by implementing one hot encoders within the traceback logic.

D. Reconfigurable Design with Dynamic Precision

The sequence alignment and edit distance calculation follow the same data flow of forward cell updating. The difference between alignment and edit distance calculation is the used scoring function. The scoring function of edit distance computation normally requires lower data bit width than alignment workloads. RAPIDx is reconfigurable to support these two workloads by adopting two types of PIM precisions. Moreover, we leverage the precision difference to further improve the performance of edit distance calculation.

1) Alignment Computation

For different alignment tools and target genomes to be aligned, various scoring functions with affine gap penalties may be applied. For example, BWA-MEM [9] uses a matching score $A = 1$, mismatch penalty $B = 4$, gap open penalty $o = 6$, and gap extension penalty $e = 1$. The other popular alignment tool, Minimap2 [7], uses a default scoring function with $A = 2, B = 4, o = 4, e = 2$. According to Section IV, the minimum data width should satisfy $\lceil \log_2(M+2o+2e+1) \rceil$. For most scoring functions with affine gap penalties, a 5-bit PIM precision is able to realize accurate alignment without overflow.

2) Edit Distance Calculation

Edit distance (or Levenshtein distance) is a metric to measure the minimum number of deletion, insertion, and substitution required to transform one string to the other one. Edit distance calculation can be regarded as a simplified version of sequence alignment, where the matching score is 0 while mismatch/gap opening/gap extension penalties are all 1. $\lceil \log_2(M + 2o + 2e + 1) \rceil = 3$-bit data width provides sufficient precision for edit distance calculation. Therefore, RAPIDx decreases the arithmetic precision from 5-bit to 3-bit when computing edit distance. This is beneficial to further improve throughput and reduce energy dissipation.

RAPIDx realizes the switching between the mentioned two types of PIM precisions through issuing different sets of commands to CMs. The commands for 3-bit and 5-bit precisions differ in they activate and access different ReRAM rows in CM to realize different computing precisions. So the overhead of PIM precision switching is negligible.

VI. Evaluation

A. Experimental Setup

Methodology: We use VTEAM [49] with $R_{OFF} = 300k$ and $R_{ON} = 10k$ to model ReRAM cell. The other parameters are same with [46] that align with the practical ReRAM device [54]. The energy consumption and latency of PIM operations in RAPIDx are measured based on 10,000 Monte Carlo simulations in SPICE. The operation voltage of PIM is V_0=1V, and the worst-case switching latency is 2ns. The hardware parameters of ReRAM subarray are obtained from NVSim [55]. Its peripheral circuits, including shifter, interleaved bit-serial max finder, and traceback logic, are implemented using *Verilog* and synthesized by *Synopsys Design Compiler* on 45nm process node [56]. The area and energy consumption of sequence buffer are estimated using CACTI [57]. RAPIDx's frequency is set to 500MHz, matching the switching time of ReRAM device. We also develop a in-house simulator to estimate the DNA alignment performance and energy consumption.

RAPIDx Configurations: Total 64 tiles are implemented in RAPIDx and each RAPIDx tile has 2MB size, containing one CM and 15 TBMs. Each ReRAM subarray consists of 1024×1024 cells and the width of column MUX output is 128-bit. The parameter selection is discussed in Section VI-C. The arithmetic precision is set to 5-bit for sequence alignment and 3-bit for edit distance calculation, which avoids overflow and maximizes the performance.

Table II: Error rates of generated datasets

Type	Substitution	Insertion	Deletion	Total
PacBio	1.5%	9.0%	4.5%	15%
ONT_2D	16.5%	5.0%	8.5%	30%
Illumina	3%	1%	1%	5%

Datasets: We test RAPIDx's performance on both short and long reads. The sequence length of short reads ranges from 100bp to 500bp while the long reads vary from 2kbp to 10kbp. We use the homologous chromosomes, GRCh38 [58], from the National Center for Biotechnology Information (NCBI). The chromosomes, including 1 to 22, X, and Y, are used and the unmapped contigs are removed. These chromosomes contain 3 billion bp in total. The available memory space in RAPIDx is not able to store the entire genome. We assume RAPIDx fetches the query and reference sequences from the host memory for alignment.

Table III: Hardware specifications of CPU and GPU baselines

CPU	Intel Xeon E5-2680 12 cores / 24 threads / 2.5GHz	**GPU**	**Geforce GTX 1080 Ti**
Cache	L1/L2/L3: 32KB/256KB/30MB	**Frequency**	1582 MHz
Memory	256GB / DDR4-2133MHz	**Memory**	11GB GDDR5X
TDP	120W	**TDP**	250 W

As Table II, we generate the long-read data (PacBio and ONT datasets) using the sequence read simulator PBSIM [59]. PacBio and ONT have 15% and 30% error rate, respectively. PBSIM's default error profile and continuous long read (CLR) mode are used. The short-read Illumina datasets are produced by Mason [60] with 5% error rate. Both RAPIDx and other baselines are tested using at least 100,000 reads for each length.

Table IV: Specifications of ASIC baselines

Design	**ABSW [11]**	**GenASM [11]**
Specifications	40nm with 480MHz frequency	28nm with 1GHz frequency
	Area: 5.51mm^2, Power: 1.2W	Area: 10.69mm^2, Power: 3.2W

Baselines: We compare the alignment performance of RAPIDx with state-of-the-art CPU, GPU, PIM, and ASIC accelerators. The CPU baselines include two libraries developed using C++, Minimap2 [7] and Edlib [6]. Minimap2 utilizes banded DP algorithms with affine gap penalties and adopts SIMD and multithreading to maximize the performance. Edlib is a C++ program that makes use of edit distance and Myers's bit-vector algorithm [34] to parallelize the alignment and

distance computation. The GPU baseline, GASAL2 [8], is optimized for GPU and delivers high throughput on various alignment workloads. We compile and run the programs on a server with hardware specifications in Table III. The other parameters are the same as the original papers [6]–[8] without explicit specifications. We compare RAPIDx with four PIM designs, including RAPID [28], AlignS [26], AligneR [27], and PIM-Aligner [36]. We also compare RAPIDx with two ASIC accelerators, ABSW [11] and GenASM [12]. Their hardware configurations are given in Table IV.

B. Algorithm Validation

The bandwidth of adaptive banded DP alignment is key to the alignment accuracy and efficiency. The base bandwidth w in the bandwidth calculation function $B = \min(w + 0.01 \times L, 100)$ determines the resulted bandwidth for sequence length L. Large w guarantees high alignment accuracy but increases the required computation and memory complexity.

Table V: Alignment accuracy of banded DP algorithms

Read Type	Adaptive Wavefront	Base bandwidth w				
		10	20	30	40	50
Short Read (Illumina)	No	100.0%	100.0%	100.0%	100.0%	100.0%
	Yes	100.0%	100.0%	100.0%	100.0%	100.0%
Long Read (ONT_2D)	No	6.51%	39.69%	31.33%	61.44%	71.13%
	Yes	99.23%	99.64%	99.85%	99.85%	99.95%

We perform Monte Carlo simulations to validate the accuracy of adaptive banded parallelized DP alignment using different parameters. The alignment results of original DP with affine gap penalty in Eq (1) are regarded as the ground truth. Both of the tested algorithm adopt the identical scoring function $A = 2, B = 4, o = 4, e = 2$ with Minimap2 [7]. We randomly sample 1,000,000 short and long sequence reads from the read simulator. Illumina and ONT_2D in Table II are adopted as the reading scheme for short reads and long reads, respectively.

Table V gives the alignment accuracy, where the base bandwidth w is ranging from 10 to 50 and the bandwidth is calculated by $B = \min(w+0.01\times L, 100)$. We also add another dimension that enables or disables the adaptive wavefront direction. The results show that the accuracy for short read is all 100% even without adaptive wavefront direction. This is because Illumina only incurs 5% error. For long reads, the algorithm without adaptive wavefront direction yields unsatisfactory accuracy. Increasing w to 50 only yields 71.13% accuracy. This is because ONT_2D has lower reading quality, making the optimal alignment path more likely to be away from the diagonal. The fixed wavefront direction is unable to track and cover the optimal path. After enabling adaptive wavefront direction, a base bandwidth w of 10 achieves 99.23% accuracy. It is observed that the optimal w varies for reading schemes and sequence lengths. To balance alignment efficiency and accuracy, we choose $w = 10$ for short reads and $w = 30$ for long reads, which incurs 0.15% accuracy degradation.

C. Design Space Exploration

1) ReRAM Subarray Size

The ReRAM subarray size determines the memory density. The parasitic wire resistance is a major factor limiting the ReRAM size [46]. To study the impact of non-ideal wire resistance, we use the same model in [46] and assume the unit wire resistance between row or column is $R_w = 10\Omega$. The upper bound and lower bound of three critical voltages (operation voltage V_0, isolation voltages V_{HS} and V_{VS}) under different ReRAM array size are depicted in Figure 8. It shows the used $V_0 = 1.0$V falls in the allowed value range when array size is 1024×1024. The effective ranges for voltages V_{HS} and V_{VS} show we can set the isolation voltages to $V_{HS} = 0.2V, V_{VS} = 1.0V$ to satisfy the constraints for size 1024×1024. Given these results, the wire resistance does not affect the correct functionality of RAPIDx under ReRAM array size 1024×1024. This is because: 1. RAPIDx uses 2-input PIM operation to perform alignment, reducing the effects of wire resistance. 2. The 10kΩ R_{ON} is $10\times$ larger than [46], making RAPIDx receive less impact from the wire resistance. Meanwhile, the chip-verified ReRAM [45] with 1024 dimension also demonstrates that the ReRAM subarray in RAPIDx is practical to manufacture.

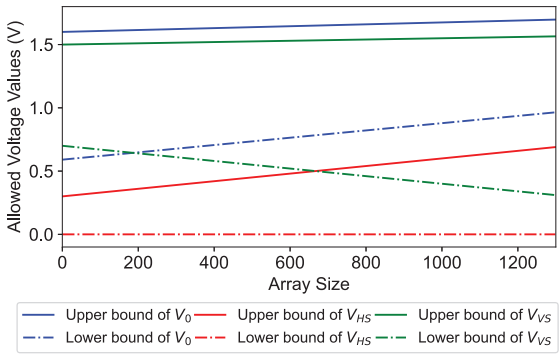


Figure 8: The lower bound and upper bound of voltages V_0, V_{HS}, V_{VS} under different ReRAM array sizes.

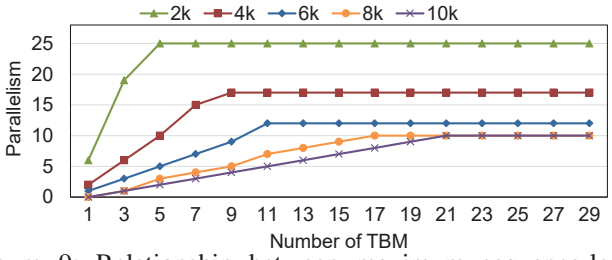
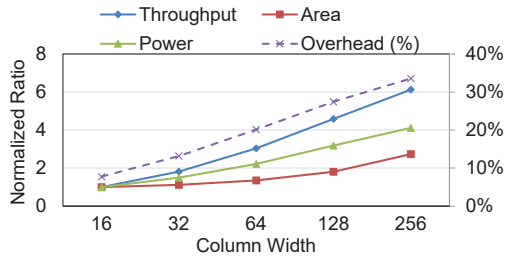


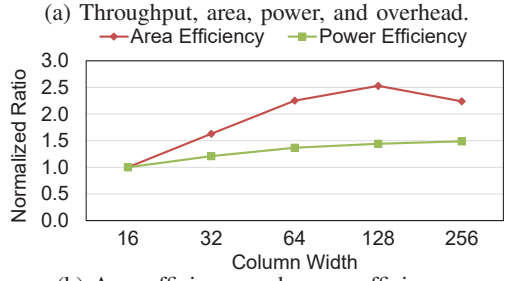
Figure 9: Relationship between maximum sequence-level parallelism and number of TBMs on long reads.

2) Number of TBMs in Each Tile

The memory complexity of alignment is dominated by traceback data storage because the traceback data for a batch of sequences need to be stored until all DP alignment steps are finished. Therefore, each CM can access the memory space of t TBMs. The number of TBMs in each tile determines the supported maximum sequence length of RAPIDx. Each TBM is a 1024×1024 ReRAM subarray, thus each TBM can store $\frac{1024^2}{2}$ points of traceback data, where 2 denotes the 2-bit traceback information. Considering the sequence alignment or edit distance calculation has a bandwidth B and sequence length m, the number of TBMs t in each tile, satisfies $m \leq \frac{1024^2}{2B}t$. However, the memory requirement increases linearly by $k\times$ when each CM processes k sequences in parallel. In this case, the maximum sequence level parallelism (or the memory segment) becomes $k \leq \lfloor \frac{1024^2}{2m \cdot B}t \rfloor$. On the other hand, k will not exceed the maximum segment number in each ReRAM subarray $k \leq \lfloor \frac{1024}{B} \rfloor$. Therefore, the relationship between number of TBMs t, sequence-level parallelism k, and sequence length m



(a) Throughput, area, power, and overhead.



(b) Area efficiency and power efficiency.

Figure 10: Performance comparison for different column widths of peripheral circuits.

is given by $k \leq \min(\lfloor \frac{1024}{B} \rfloor, \lfloor \frac{1024^2}{2m \cdot B} t \rfloor)$.

The sequence-level parallelism under various sequence lengths and TBM numbers is given in Fig. 9. Shorter sequences require less TBMs to achieve the maximum parallelism. The $k_{\max}$ of sequences longer than 8kbp is limited by $\lfloor \frac{1024}{B} \rfloor$. As the maximum value of B is 100, $\lfloor \frac{1024}{B} \rfloor \leq 10$ for sequences over 8kbp. In this case, the number of TBMs t, making $\lfloor \frac{1024^2}{2m \cdot B} t \rfloor > 10$, can not further improve the performance. We implement $t = 15$ TBMs to ensure sufficient sequence-level parallelism for 10kbp while balancing area overhead. Thus, each RAPIDx tile consists of 16 ReRAM subarrays.

3) Column Width of Peripheral Circuits

The peripheral circuits of CM are connected to the column MUX of SA and have the same width as column MUX. The column width of peripheral circuits is a design parameter affecting the overall throughput, power, and area. Fig. 10 shows the comparison of performance for different widths (from 16 to 256) of peripheral circuits. As shown in Fig. 10a, wider column width leads to higher throughput and the increasing trend of throughput is slightly more significant than area and power when the width is between 16 and 128. The overhead here denotes the percentage of peripheral circuits area to single ReRAM subarray. We depict the area efficiency and power efficiency in Fig. 10b to understand the relationship between efficiency and column width. Area efficiency and power efficiency peak at width 128 and 256, respectively. However, wider width introduces larger area overhead to CM. We choose the column width of 128 to achieve good tradeoff between efficiency and overhead.

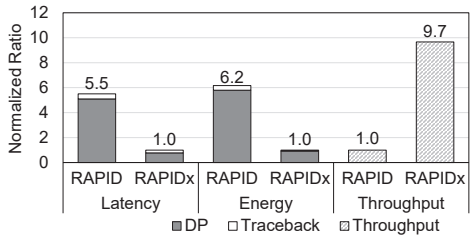
D. Area and Power Results

The area and power breakdown of RAPIDx is summarized in Table VI. The bit-serial max finder takes up 62.3% area and 61.6% power of the peripheral circuits, respectively. About 16% area of CM is consumed by peripheral circuits. Each RAPIDx tile is composed of 1 CM and 15 TBMs, consuming 0.0.637mm^2 area and 0.16W power. We measure the power dissipation of RAPIDx under sequence alignments for long sequence lengths (2kbp to 10kbp) with enabling the traceback procedure. As a result, the area and power of RAPIDx with 64 tiles in total are 40.8mm^2 and 10.3W, respectively.

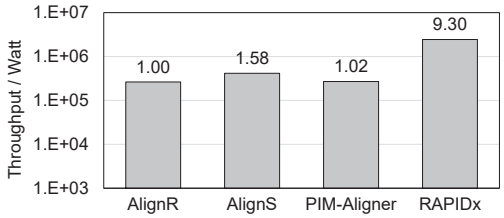
Table VI: Area and power breakdown of RAPIDx

Peripheral Circuits	Area (um^2)	Power (mW)
Shifter	542.6	0.03
Max Finder	4, 520.8	2.05
Traceback Logic	1, 872,4	1.21
Others	325.2	0.03
Total	7, 260.9	3.32
Seq. Buffer	8, 492.6	1.5
ReRAM Subarray	38, 395.0	9.76

RAPIDx	Area	Power
Per tile	637,334.4um^2	0.16W
Total	40.8mm^2	10.3W



(a) Latency, energy, and throughput comparison of RAPID [28] and RAPIDx.



(b) Energy efficiency comparison (in log scale) with PIM-based accelerators: AlignS [26], AligneR [27], and PIM-Aligner [36].

Figure 11: Comparison with four PIM baselines, RAPID [28], AlignS [26], AligneR [27], and PIM-Aligner [36].

E. Performance Evaluation

We measure the performance of RAPIDx on various sequence lengths and compare with state-of-the-art acceleration solutions for genome sequence analysis. The sequences are divided into short reads (<1kbp) and long reads (>1kbp). Two types of workloads are considered, including sequence alignment in Section VI-E1 VI-E2 VI-E3 and edit distance calculation in Section VI-E4. RAPIDx uses 5-bit integer for alignment and 3-bit integer for edit distance calculation.

1) Comparison with PIM Designs

Our previous work, RAPID [28], is also a ReRAM-based PIM design for sequence alignment. First, we evaluate the reduction of processing latency and energy by adopting the parallelized DP alignment. The comparison of latency and energy with the original DP alignment for a single step of cells updating is shown in Fig. 11 (a). RAPID uses the unoptimized DP alignment with 32-bit precision. The used PIM operations are the same as RAPIDx. As a result, the parallelized DP alignment based on difference presentation yields 5.5× latency reduction and 6.2× energy reduction over the original DP alignment. The latency and energy consumed by forward DP computation are reduced by 82% and 84% over the previous RAPID, respectively. The gain comes from the reduced arithmetic precision from 32-bit to 5-bit as well as the

parallelized computation. On the other hand, the reduction of latency and energy for traceback is less significant. Although the parallelized DP alignment requires less bit width, its traceback is more complicated and involves more computations than the original DP algorithm. The longest sequence support by RAPIDx is 10kbp so we test the throughput of RAPID and RAPIDx on this length in Fig. 11 (a). RAPIDx yields $9.7\times$ throughput improvement over RAPID due to the low complexity and high data parallelism provided by adaptive banded parallelized DP alignment.

In Fig. 11 (b), we compare the energy efficiency with the other three PIM designs for short-read alignment, including AlignS [26], AligneR [27], and PIM-Aligner [36]. The read length is 100bp and the alignment efficiency is measured by the alignment throughput (reads per second) divided by the power dissipation. RAPIDx delivers $5.9\times$ to $9.3\times$ alignment efficiency compared to other PIM designs. It should be also noted that the area of mentioned PIM designs is: RAPIDx (40.8mm^2), AlignR (36.1mm^2), AlignS (62.5mm^2), and PIM-Aligner (59.3mm^2). This shows that RAPIDx achieves $8.4\times$ to $13.3\times$ throughput/W/mm^2 efficiency compared to other designs. This is because the optimized adaptive banded parallelized DP alignment in RAPIDx significantly reduces computational complexity over the original full DP algorithm and allows to fully exploit the internal data parallelism of ReRAM. In comparison, AlignS, AligneR, and PIM-Aligner realize alignment based on FM-index algorithm, which requires multiple steps of computation and incurs data dependency [61]. AlignS, AligneR, and PIM-Aligner only support fixed read length while RAPIDx supports both short reads and long reads, making RAPIDx more scalable and reconfigurable.

2) Performance Comparison on Short-read Alignment

For alignment tasks on short reads, the length ranges from 100bp to 250bp and we use Minimap2 [7] as the CPU baseline and GASAL2 [8] as the GPU baseline. Fig. 12 depicts the alignment throughput of RAPIDx, Minimap2, and GASAL2 for short reads in log scale. The alignment throughputs for three tested accelerators slightly decrease as the sequence length grows. RAPIDx on average delivers $131.1\times$ and $46.8\times$ throughput over Minimap2 and GASAL2, respectively. The processing latency of RAPIDx is longer than the other two counterparts due to the fact that a single PIM operation of RAPIDx requires longer latency than CPU and GPU. However, the row-parallel PIM operations provide higher computation parallelism. The proposed multi-level parallelism scheme ensures multiple reference and query sequences can be aligned in parallel, significantly increasing the data parallelism and PIM utilization. As a result, RAPIDx achieves an average throughput of 13.9M reads/s for short-read alignment.

DP alignment is computation-intensive and the bottleneck of CPU is the limited computing cores. Even though GPU has much more computing capabilities than CPU, we observe that GASAL2 only yields $2.4\times$ to $3.6\times$ speedup over Minimap2 because Minimap2 uses a banded DP algorithm and multi-threading to reduce the complexity, thus improving the overall throughput. In comparison, GASAL2 requires more computing resources since it does not finely optimize the original DP alignment. RAPIDx is an algorithm and hardware co-optimization

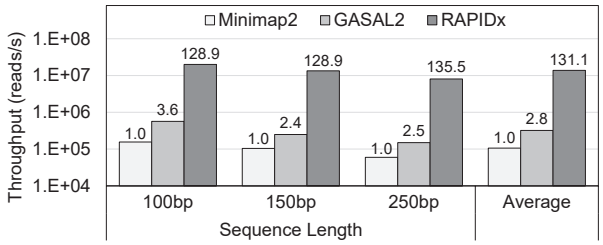


Figure 12: Alignment throughput comparison of RAPIDx, GASAL2 [8], and Minimap2 [7] for short reads.

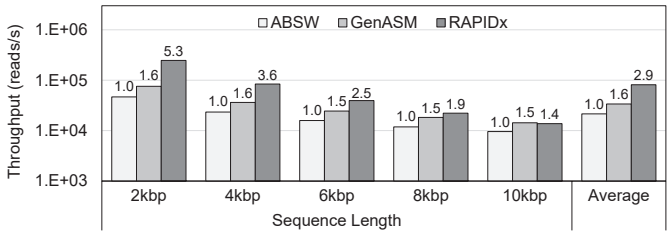


Figure 13: Alignment throughput comparison of GenASM [12], ABSW [11], and RAPIDx for long reads.

that addresses the deficits of Minimap2 and GASAL2.

3) Performance Comparison on Long-read Alignment

For long reads from 2kbp to 10kbp, ABSW [11] and GenASM [12], are adopted as the two ASIC baselines. The throughput comparison with ASIC for long-read alignment is shown in Fig. 13, where the performance of ASIC baselines is scaled to 45nm process for the fair comparison. RAPIDx achieves the highest throughput with an average speedup of $2.9\times$ and $1.8\times$ over ABSW and GenASM, respectively. Due to the limited on-chip memory space, both ABSW and GenASM are not able to store the entire traceback matrix for long reads. They rely on large off-chip memory to store the intermediate data. To realize alignment for long sequences, they use the overlapping scheme [13] to divide the long sequence into short chunks and the neighbor chunks are overlapped. ABSW and GenASM need to consecutively process the short chunks. As a result, the overlapping area incurs additional computational complexity, which degrades the performance.

ABSW and RAPIDx are based on banded DP algorithms. The difference between this work and ABSW is RAPIDx adopts the optimized 5-bit parallelized DP alignment based on difference representations. ABSW uses 12-bit precision to ensure arithmetic precision for DP alignment. RAPIDx's lower bit width reduces both the complexity and the memory footprint of DP alignment compared to ABSW. The other limitation of ABSW is it can only process a fixed bandwidth of 128 since a total of 128 processing elements (PEs) are implemented and dedicated to updating the wavefront of banded alignment. This means ABSW is only able to align one sequence at a time. In contrast, RAPIDx accepts a batch of sequences and distributes them into different tiles to perform alignment in parallel.

4) Performance Comparison on Edit Distance Computation

To evaluate the performance of edit distance calculation, we compare RAPIDx with Edlib [6] on three lengths (100bp, 1kbp, and 10kbp). Fig. 14 shows the throughput of RAPIDx and Edlib with or without traceback process. Knowing the edit distance of two sequences is enough for some scenarios,

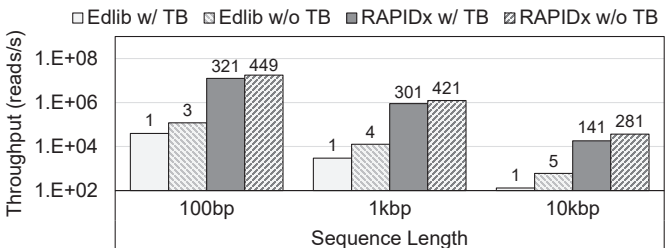


Figure 14: Throughput and latency comparison of RAPIDx and Edlib [6] for edit distance computation.

without the need for traceback process. So we test the cases with or without traceback. The throughput of RAPIDx with traceback is $141\times$ to $321\times$ over Edlib. After disabling the traceback, the speedup of RAPIDx is less significant. $56\times$ to $149\times$ improvements of RAPIDx are observed compared to Edlib. Although Edlib adopts optimized Myers's bit-vector algorithm [34] with banded alignment to increase computation efficient, it is a single-thread program only able to access limited computing resources of CPU. Hence, the performance dramatically decreases after enabling traceback.

F. Discussions

Host-RAPIDx System Design: RAPIDx is a PIM-based domain-specific accelerator and works as the domain-specific co-processor for speeding up computation-intensive genome sequence alignments. We consider a system that transfers data between RAPIDx and the host. The sequencing and configuration data are sent from the host to RAPIDx. We estimate the memory bandwidth required by RAPIDx and the results show that required memory bandwidth decreases when sequence length grows. The required peak memory bandwidth is 1.41GB/s at 100bp. For the host side, the popular DDR4 Dual-Inline Memory Module (DIMM) that provides over 12.8GB/s data rate can easily satisfy the bandwidth requirement. The other consideration is the processing latency. As pointed out in Section VI-E, RAPIDx requires longer latency than CPU. Considering that genome sequence alignment is not a latency-sensitive task, the long latency will not become a major factor that limits system performance. Hence, RAPIDx can be integrated into existing computer machines with negligible hardware modifications.

Flexible Scoring Functions: The affine gap penalty of DP alignment will be changed according to different application scenarios. RAPIDx is able to flexibly support various scoring functions. When the gap open penalty o equals the gap extension penalty e, the affine gap penalty becomes a linear gap penalty scoring. If $e = 0$, RAPIDx implements a constant gap penalty where only opening a gap leads to a penalty, discouraging the number of gaps but tends to result in long gaps. Whereas, if $o \neq e$ and both of o and e are non-zero values, we have affine gap penalty, which is the widely used gap penalty model for DNA alignment. The affine gap penalty tries to align the given sequences with fewer and smaller gaps as compared to the constant gap penalty. No architectural and data flow modifications need to be made to RAPIDx if we want to switch between different scoring functions. The support for flexible scoring is realized by setting associated constant values into the intermediate data rows of CM before alignment.

ReRAM's Write Endurance: ReRAM cell has limited write endurance, so RAPIDx will fail after exceeding the endurance limit. As shown in Fig. 6 (c), the wavefront alignment at each iteration needs to write the rows in the computing region once. Fig. 4 shows the required number of iterations equals to the sum of reference and query sequences' lengths. We can apply wear leveling techniques to reduce the imbalance effect, thus extending the write endurance of ReRAM. The wear leveling is realized via moving the computing region over the row dimension. Specifically, this can be done through changing the writing address without additional overhead. Moreover, we observe some ReRAM devices [62] provide 10^{12} write endurance. In this case, RAPIDx can align over 10^{14} sequences with length 150bp. We notice that one of the most advanced next-generation sequencing (NGS) platforms from Illumina, *NextSeq 1000 & 2000*, generates a maximum 1.2 billion reads (each has a length of 150bp) in 11 to 48 hours [63]. Therefore, each RAPIDx is able to support the alignment task of each NGS sequencer for at least 100 years.

VII. Conclusion

In this work, we propose a novel PIM accelerator, RAPIDx, for sequence alignment. We leverage the parallelized DP algorithm using difference representation to reduce the required data width from 32-bit to 5-bit integers. Based on this, we propose adaptive banded parallelized DP alignment to adaptively adjust the bandwidth and wavefront direction, reducing the quadratic complexity to near-linear complexity while only incurring 0.15% accuracy degradation. Then we present the PIM architecture on ReRAM that exploits four-level data parallelism to efficiently implement the proposed algorithm. We develop peripheral circuits and row-parallel PIM data flow to support in-situ alignment with low latency. The evaluation results demonstrate that RAPIDx provides $131.1\times$ and $46.8\times$ better short-read alignment throughput compared to CPU and GPU baselines, respectively. For long-read alignment, RAPIDx delivers up to $2.9\times$ and $9.3\times$ throughput improvements compared to state-of-the-art ASIC and PIM accelerators.

Acknowledgment

This work was supported in part by CRISP, one of six centers in JUMP (an SRC program sponsored by DARPA), SRC Global Research Collaboration (GRC) grant, and NSF grants #1826967, #1911095, #2003279, #2112665, #2112167, and #2100237.

The authors thank the anonymous reviewers from JETC and the constructive discussion with Behnam Khaleghi to help improve the quality of this work.

References

[1] F. E. Dewey *et al.*, "Dna sequencing: clinical applications of new dna sequencing technologies," *Circulation*, vol. 125, no. 7, pp. 931–944, 2012.

[2] A. J. Drummond and A. Rambaut, "Beast: Bayesian evolutionary analysis by sampling trees," *Evolutionary Biology*, vol. 7, no. 1, pp. 1–8, 2007.

[3] S. J. Watson *et al.*, "Viral population analysis and minority-variant detection using short read next-generation sequencing," *Philosophical Transactions of the Royal Society B: Biological Sciences*, vol. 368, no. 1614, p. 20120205, 2013.

[4] S. B. Needleman and C. D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," *JMB*, vol. 48, no. 3, pp. 443–453, 1970.

[5] T. F. Smith *et al.*, "Identification of common molecular subsequences," *JMB*, vol. 147, no. 1, pp. 195–197, 1981.

[6] M. Šošić and M. Šikić, "Edlib: a c/c++ library for fast, exact sequence alignment using edit distance," *Bioinformatics*, vol. 33, no. 9, pp. 1394–1395, 2017.
[7] H. Li, "Minimap2: pairwise alignment for nucleotide sequences," *Bioinformatics*, vol. 34, no. 18, pp. 3094–3100, 2018.
[8] N. Ahmed *et al.*, "Gasal2: a gpu accelerated sequence alignment library for high-throughput ngs data," *Bioinformatics*, vol. 20, no. 1, pp. 1–20, 2019.
[9] H. Li and R. Durbin, "Fast and accurate long-read alignment with burrows–wheeler transform," *Bioinformatics*, vol. 26, no. 5, pp. 589–595, 2010.
[10] B. Langmead and S. L. Salzberg, "Fast gapped-read alignment with bowtie 2," *Nature methods*, vol. 9, no. 4, pp. 357–359, 2012.
[11] Y.-L. Liao *et al.*, "Adaptively banded smith-waterman algorithm for long reads and its hardware accelerator," in *International Conference on Application-specific Systems, Architectures and Processors*, 2018, pp. 1–9.
[12] D. S. Cali *et al.*, "Genasm: A high-performance, low-power approximate string matching acceleration framework for genome sequence analysis," in *IEEE/ACM MICRO*, 2020, pp. 951–966.
[13] Y. Turakhia *et al.*, "Darwin: A genomics co-processor provides up to 15,000× acceleration on long read assembly," in *ASPLOS*, 2018.
[14] E. F. de Oliveira Sandes *et al.*, "Cudalign 4.0: Incremental speculative traceback for exact chromosome-wide alignment in gpu clusters," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 10, pp. 2838–2850, 2016.
[15] J. Arram *et al.*, "Leveraging fpgas for accelerating short read alignment," *IEEE/ACM TCBB*, vol. 14, no. 3, pp. 668–677, 2017.
[16] "Dna sequencing costs: Data from the nhgri genome sequencing program (gsp)," www.genome.gov/sequencingcostsdata.
[17] "Genbank and wgs statistics," https://www.ncbi.nlm.nih.gov/genbank/statistics/.
[18] A. M. Wenger *et al.*, "Accurate circular consensus long-read sequencing improves variant detection and assembly of a human genome," *Nature biotechnology*, vol. 37, no. 10, pp. 1155–1162, 2019.
[19] M. Gokhale *et al.*, "Processing in memory: The terasys massively parallel pim array," *Computer*, vol. 28, no. 4, pp. 23–31, 1995.
[20] J. Ahn *et al.*, "Pim-enabled instructions: a low-overhead, locality-aware processing-in-memory architecture," in *ISCA*, 2015, pp. 336–348.
[21] S. Li *et al.*, "Pinatubo: a processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories," in *DAC*, 2016, p. 173.
[22] S. Gupta *et al.*, "Nnpim: A processing in-memory architecture for neural network acceleration," *IEEE Transactions on Computers*, vol. 68, no. 9, pp. 1325–1337, 2019.
[23] R. Kaplan *et al.*, "A resistive cam processing-in-storage architecture for dna sequence alignment," *IEEE Micro*, vol. 37, no. 4, pp. 20–28, 2017.
[24] ——, "Bioseal: In-memory biological sequence alignment accelerator for large-scale genomic data," in *ACM International Systems and Storage Conference*, 2020, pp. 36–48.
[25] W. Huangfu, S. Li, X. Hu, and Y. Xie, "Radar: a 3d-reram based dna alignment accelerator architecture," in *DAC*, 2018, pp. 1–6.
[26] S. Angizi *et al.*, "Aligns: A processing-in-memory accelerator for dna short read alignment leveraging sot-mram," in *DAC*, 2019, pp. 1–6.
[27] F. Zokaee *et al.*, "Aligner: A process-in-memory architecture for short read alignment in rerams," *IEEE Computer Architecture Letters*, vol. 17, no. 2, pp. 237–240, 2018.
[28] S. Gupta *et al.*, "Rapid: A reram processing in-memory architecture for dna sequence alignment," in *IEEE/ACM ISLPED*, 2019, pp. 1–6.
[29] K. Liu *et al.*, "Barking up the wrong treelength: the impact of gap penalty on alignment and tree accuracy," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 6, no. 1, pp. 7–21, 2008.
[30] D. Fujiki *et al.*, "Seedex: A genome sequencing accelerator for optimal alignments in subminimal space," in *IEEE/ACM MICRO*, 2020, pp. 937–950.
[31] K.-M. Chao *et al.*, "Aligning two sequences within a specified diagonal band," *Bioinformatics*, vol. 8, no. 5, pp. 481–487, 1992.
[32] S. Gupta *et al.*, "Felix: Fast and energy-efficient logic in memory," in *IEEE/ACM ICCAD*, 2018, pp. 1–7.
[33] M. Burrows and D. Wheeler, "A block-sorting lossless data compression algorithm," in *Digital SRC Research Report*, 1994.
[34] G. Myers, "A fast bit-vector algorithm for approximate string matching based on dynamic programming," *Journal of the ACM (JACM)*, vol. 46, no. 3, pp. 395–415, 1999.
[35] H. Suzuki and M. Kasahara, "Introducing difference recurrence relations for faster semi-global alignment of long sequences," *Bioinformatics*, vol. 19, no. 1, pp. 33–47, 2018.
[36] S. Angizi *et al.*, "Pim-aligner: A processing-in-mram platform for biological sequence alignment," in *DATE*, 2020, pp. 1265–1270.
[37] S. F. Altschul *et al.*, "Basic local alignment search tool," *Journal of Molecular Biology*, vol. 215, no. 3, pp. 403–410, 1990.
[38] D. J. Lipman and W. R. Pearson, "Rapid and sensitive protein similarity searches," *Science*, vol. 227, no. 4693, pp. 1435–1441, 1985.
[39] S. S. Banerjee *et al.*, "Asap: Accelerated short-read alignment on programmable hardware," *IEEE Transactions on Computers*, 2019.
[40] O. Gotoh, "An improved algorithm for matching biological sequences," *Journal of molecular biology*, vol. 162, no. 3, pp. 705–708, 1982.
[41] K. Lee *et al.*, "Bit parallel 6t sram in-memory computing with reconfigurable bit-precision," in *DAC*, 2020, pp. 1–6.
[42] J. Boukhobza *et al.*, "Emerging nvm: A survey on architectural integration and research challenges," *ACM Transactions on Design Automation of Electronic Systems*, vol. 23, no. 2, pp. 1–32, 2017.
[43] D. Reis *et al.*, "Computing in memory with fefets," in *ISLPED*, 2018, pp. 1–6.
[44] M. Kim *et al.*, "An embedded nand flash-based compute-in-memory array demonstrated in a standard logic process," *IEEE Journal of Solid-State Circuits*, vol. 57, no. 2, pp. 625–638, 2021.
[45] C.-X. Xue *et al.*, "A 22nm 4mb 8b-precision reram computing-in-memory macro with 11.91 to 195.7 tops/w for tiny ai edge devices," in *ISSCC*, vol. 64, 2021, pp. 245–247.
[46] N. Talati *et al.*, "Logic design within memristive memories using memristor-aided logic (magic)," *IEEE Transactions on Nanotechnology*, vol. 15, no. 4, pp. 635–650, 2016.
[47] J. Borghetti *et al.*, "Memristive switches enable stateful logic operations via material implication," *Nature*, vol. 464, no. 7290, pp. 873–876, 2010.
[48] B. C. Jang *et al.*, "Memristive logic-in-memory integrated circuits for energy-efficient flexible electronics," *Advanced Functional Materials*, vol. 28, no. 2, p. 1704725, 2018.
[49] S. Kvatinsky *et al.*, "Vteam: a general model for voltage-controlled memristors," *IEEE TCAS II*, vol. 62, no. 8, pp. 786–790, 2015.
[50] A. Haj-Ali *et al.*, "Efficient algorithms for in-memory fixed point multiplication using magic," in *IEEE ISCAS*, 2018, pp. 1–5.
[51] S. Kvatinsky *et al.*, "MAGIC – memristor-aided logic," *TCAS II*, vol. 61, no. 11, 2014.
[52] M. Imani *et al.*, "Floatpim: In-memory acceleration of deep neural network training with high precision," in *ISCA*, 2019.
[53] H. Suzuki and M. Kasahara, "Acceleration of nucleotide semi-global alignment with adaptive banded dynamic programming," *BioRxiv*, p. 130633, 2017.
[54] J. J. Yang *et al.*, "Memristive devices for computing," *Nature nanotechnology*, vol. 8, no. 1, pp. 13–24, 2013.
[55] X. Dong *et al.*, "Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *IEEE TCAD*, vol. 31, no. 7, pp. 994–1007, 2012.
[56] J. E. Stine *et al.*, "Freepdk: An open-source variation-aware design kit," in *IEEE International Conference on Microelectronic Systems Education*, 2007, pp. 173–174.
[57] N. Muralimanohar *et al.*, "Cacti 6.0: A tool to model large caches," *HP laboratories*, vol. 27, p. 28, 2009.
[58] N. C. for Biotechnology Information, "Genome reference consortium human build 38," https://www.ncbi.nlm.nih.gov/assembly/GCF_000001405.26, 2013.
[59] Y. Ono *et al.*, "Pbsim: Pacbio reads simulator—toward accurate genome assembly," *Bioinformatics*, vol. 29, no. 1, pp. 119–121, 2013.
[60] M. Holtgrewe, "Mason–a read simulator for second generation sequencing data," *Technical Report FU Berlin*, 2010.
[61] W. Huangfu *et al.*, "Medal: Scalable dimm based near data processing accelerator for dna seeding algorithm," in *IEEE/ACM MICRO*, 2019, pp. 587–599.
[62] Q. Luo *et al.*, "Nb_{1-x} o2 based universal selector with ultra-high endurance ($> 10^{12}$), high speed (10ns) and excellent v_{th} stability," in *Symposium on VLSI Technology*, 2019, pp. T236–T237.
[63] "Illumina sequencing platforms," https://www.illumina.com/systems/sequencing-platforms.html.

Weihong Xu received the B.E. degree in information engineering and M.E. degree information science and engineering from Southeast University, Nanjing, China, in 2017 and 2020, respectively. He is currently pursing the Ph.D. degree with the Department of Computer Science and Engineering, University of California at San Diego, La Jolla, CA, USA.

He is a member of the System Energy Efficiency Laboratory, University of California at San Diego, where he works on emerging memory, storage and domain-specific acceleration systems. His current research interests include algorithm and hardware co-design based on near-data processing for deep learning, database, and bioinformatics applications.

Saransh Gupta received his Ph.D. degree from the University of California, San Diego, La Jolla, CA, USA in 2021. He received his B.E. (Hons) in Electrical and Electronics Engineering from Birla Institute of Technology & Science, Pilani - K.K. Birla Goa Campus in 2016 and M.S. in Electrical and Computer Engineering from University of California San Diego in 2018. His research interests include circuit, architecture, and system level aspects of emerging computing paradigms.

Niema Moshiri received his Ph.D. degree from the University of California, San Diego, La Jolla, CA, USA in 2019. He is an Assistant Teaching Professor in the Computer Science & Engineering Department at the University of California, San Diego. He works on computational biology, with a research focus on viral phylogenetics and epidemiology. He also places a heavy emphasis on teaching, namely on the development of online educational content, primarily Massive Adaptive Interactive Texts (MAITs).

Tajana Šimunić Rosing (Fellow, IEEE) received her Ph.D. degree from Stanford University, Stanford, CA, USA, in 2001. She is a Professor, a Holder of the Fratamico Endowed Chair, and the Director of System Energy Efficiency Laboratory, University of California at San Diego, La Jolla, CA, USA. From 1998 to 2005, she was a full-time Research Scientist with HP Labs, Palo Alto, CA, USA, while also leading research efforts with Stanford University, Stanford, CA, USA. She was a Senior Design Engineer with Altera Corporation, San Jose, CA, USA. She is leading a number of projects, including efforts funded by DARPA/SRC JUMP CRISP program with focus on design of accelerators for analysis of big data, DARPA and NSF funded projects on hyperdimensional computing, and SRC funded project on IoT system reliability and maintainability. Her current research interests include energy-efficient computing, cyber–physical, and distributed systems.