Efficient Memory Layout for Pre-alignment Filtering of Long DNA Reads Using Racetrack Memory

Asif Ali Khan, Fazal Hameed, Taha Shahroodi, Alex K. Jones *Fellow, IEEE*, and Jeronimo Castrillon, *Senior Member, IEEE*

Abstract—DNA sequence alignment is a fundamental and computationally expensive operation in bioinformatics. Researchers have developed *pre-alignment* filters that effectively reduce the amount of data consumed by the alignment process by discarding locations that result in a poor match. However, the filtering operation itself is memory-intensive for which the conventional Von-Neumann architectures perform poorly. Therefore, recent designs advocate compute near memory (CNM) accelerators based on stacked DRAM and more exotic memory technologies such as *racetrack memories* (RTM). However, these designs only support small DNA reads of circa 100 nucleotides, referred to as *short reads*. This paper proposes a CNM system for handling both long and short reads. It introduces a novel data-placement solution that significantly increases parallelism and reduces overhead. Evaluation results show substantial reductions in execution time ($1.32\times$) and energy consumption (50%), compared to the state-of-the-art.

1 Introduction

High-throughput sequencing (HTS) technologies produce an extensive volume of sequencing data, encompassing both *short reads* (consisting of hundreds of nucleotides) and *long reads* (extending to thousands of nucleotides). Subsequently, the generated raw data is commonly used in diverse analyses, such as disease diagnostics and genetic evolution, among others. Sequence alignment represents a fundamental and computationally demanding stage within these analysis pipelines [1].

Several algorithms are employed for sequence alignment. Dynamic programming based solutions are accurate but scale poorly with the problem size [2]. Seed-and-extension algorithms are more scalable [3], [4] but still exhaust considerable effort on genome locations that eventually do not align. Many of these poor alignment locations can be easily discarded using pre-alignment filters [5], [6], [7], [8].

Pre-alignment filters significantly reduce the number of DNA fragments that are passed on to the computationally expensive alignment phase by eliminating the apparent mismatches. This improves the end-to-end performance of the aligners by circa $2\times$ [5]. These filters employ lightweight algorithms to compute a similarity score for each candidate location. Detailed alignment operations are only performed at locations that pass a set threshold on the similarity score.

For pre-filtering to accelerate the overall processing time effectively, it must be considerably faster than the actual alignment operation. However, considering the larger genomics data sets, this can become challenging. To reduce their computational time, pre-alignment filters have been implemented in CPUs, GPUs, FPGAs, ASICs, and computing-near-memory (CNM)[1] solutions [5], [6], [7], [8], [9]. The CNM solutions dramatically reduce the data movement on the external channel between the memory and the processor.

- *Asif Ali Khan and Jeronimo Castrillon are with TU Dresden, 01069 Dresden, Germany Email: asif_ali.khan@tu-dresden.de.*
- *Fazal Hameed is with the American University of Sharjah (AUS), UAE.*
- *Alex K. Jones is with the University of Pittsburgh, 1238 Benedum Hall, 3700 O'Hara Street, Pittsburgh, PA 15261, USA.*
- *T. Shahroodi is with TU Delft, 2628 Delft, Netherlands.*

Manuscript received July XX, 2023; revised July XX, 2023.

1. This is also sometimes referred to as processing near memory (PNM) or processing in memory (PIM).

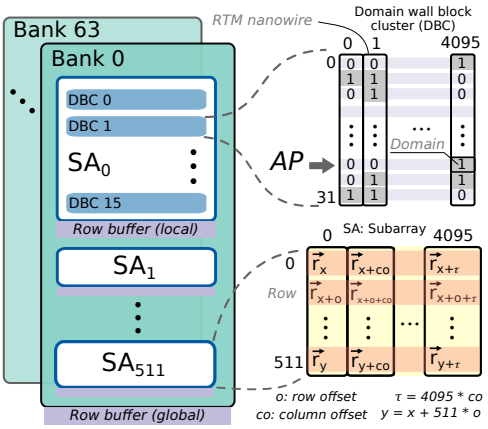

Fig. 1: Racetrack memory organization.

Pre-alignment filtering is particularly beneficial for long reads, which are predicted to revolutionize genomics [10] but require considerably more resources for processing. Unfortunately, they have received no attention from the community. SneakySnake is the only available filtering algorithm for long reads but is not compatible with CNM acceleration [8]. On the contrary, GRIM [5], ALPHA [9] and FIRM [7] are CNM-accelerated but do not support long reads.

This paper proposes an extension to the FIRM filter to support long reads. Concretely, we introduce algorithmic modifications to the FIRM filter to support long reads. Similar to FIRM [7], our CNM-accelerator leverages the unprecedented density, energy, and performance efficiencies of *racetrack memory* (RTM) [11]. However, the FIRM memory layout necessitates substantial data duplication, resulting in significant performance degradation when handling long reads. To address this challenge, we present an innovative RTM layout for our CNM accelerator that supports both short and long reads. Our layout minimizes RTM shifts to an absolute minimum and maximizes parallelism for varying numbers of parallel working bin units in the accelerator. To our best knowledge, this is the first CNM solution that efficiently handles long reads.

2 Background and Related Work

2.1 RTM Organization

RTM is a spintronic nonvolatile memory. A single cell in RTM is a magnetic nanowire, or *track* that can be split into circa

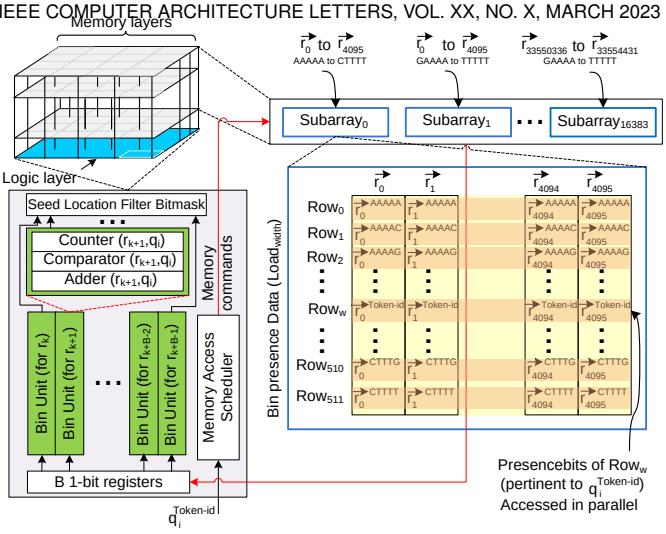

Fig. 2: GRIM filter hardware design and data mapping of 4096 bins (*i.e.*, r_0 to r_{4095}) in R to Subarray$_0$

100 magnetic regions, or *domains* separated by *domain walls* (see Fig. 1, top right). During an RTM access, a domain must be *shifted* and aligned to an access port (AP) position. RTM nanowires are typically grouped into *domain block clusters* [11] or DBCs. These DBCs are then organized into subarrays, following a similar structure to traditional memory technologies, where they form banks and ranks (left). In our proposed accelerator, each DBC comprises 4096 nanowires, with each one containing 32 domains, resulting in subarrays of dimensions 512×4096 (as illustrated in Fig. 1, bottom-right).

RTM exhibits several promising properties, including SRAM-class access speed, lower energy requirements, SRAM/DRAM-class endurance, and exceptional density. For DNA pre-alignment filtering, similar to FIRM, we favor RTM over other nonvolatile memories (NVMs) because of two primary considerations: (i) the remarkable density of RTMs, essential for accommodating entire reference genomes, and (ii) the inherently sequential memory access pattern of the application which perfectly aligns with the sequentiality in RTMs.

2.2 Pre-Alignment filtering

Pre-alignment filters proactively skip poor alignment mapping positions, minimizing the number of extend operations in read aligners [5], [7], [9]. Among others, the GRIM filter has demonstrated effective utilization of 3D-stacked memory devices, leading to significant performance improvements in read mappers through substantial parallelism [5].

GRIM-algorithm based filters [5], [7], [9] work from query genome "reads" $\mathtt{Q} = \{q_0, q_1, \ldots, q_{m-1}\}$ for alignment onto a reference genome, $\mathtt{R} = \{r_0, r_1, ..., r_{n-1}\}$, where each q_i and r_i represents a DNA read and a segment of the reference genome (called a *bin*), respectively. Each query read consists of many nucleotides from $\Gamma = \{A, C, G, T\}$ where the exact read size is determined by the sequencing technology while the bin size is typically fixed to 100 [5], [9]. Each read and bin is split into *tokens* where a token is a string consisting of t nucleotides, *i.e.*, a string in Γ^t. The reference bins are represented with binary bitvectors ($\vec{r}$), where the bit value (presence bit) at a specific position, *i.e.*, $\vec{r}[token]$ is set if *token* is present in the bin. The pre-alignment filter then compares each read with all the bins on a token-by-token basis, accumulates the bit values of $\vec{r}$ for each token in the query read, and compares the sum output of each bin with a set threshold. In other words, a bin r_k is selected for a read q_i if $\sum_{token \in q_i} \vec{r}_k[f(token)] > \mathtt{Thr}$.

2.3 State-of-the-art pre-alignment filters

Fig. 2 shows the CNM architectures employed by recent pre-alignment filters such as GRIM, ALPHA, and FIRM [5], [7], [9]. The right side of the figure shows how reference bins (r) are mapped to the memory. The accelerator compares a read q_i with B reference bins in parallel, where B is the number of bin units in the accelerator (shown in green on the left). Let us call these simultaneously executed bins a *binset*. For a fair comparison with existing filters, we assume the same binset and the memory access granularity, i.e., 4 k bits.

For each *token* in a read, the GRIM architecture loads the presence bits of a binset into the 4 k 1-bit registers of the logic layer, compares them with the read token, and updates their corresponding counters. Once all tokens of the read q_i are exhausted, the comparator compares the value of the accumulators with a given threshold and sets a bit in the filter bitmask register to 1 if the accumulator value for a given bin is greater than the threshold. The sequence aligner then reads this bitmask, discards bins having value 0, and applies a compute-intensive dynamic programming solution at other positions to find the actual mapping position.

ALPHA [9] exploits the tokens repetition in query reads to minimize the number of memory accesses in the GRIM filter. Concretely, if the set of distinct tokens in q_i is represented by Θ, *i.e.*, $\Theta(q_i) = \{\text{distinct tokens in } q_i\}$, ALPHA maintains each token count value in a dedicated buffer called *CountBuffer*. In the filtering processing, instead of accessing a row of bitvectors multiple times for repeated tokens in a read, ALPHA only accesses rows storing unique tokens to compute the sum value. For instance, for a particular bin $r_k \in \mathtt{R}$, the accumulated sum is computed as $\sum_{j \in \Theta(q_i)} \text{CountBuffer}(j) \cdot \vec{r}_k[f(j)]$.

FIRM replaces DRAM with RTM in the ALPHA design and proposes layout optimizations to minimize the inherent shift operations of the RTM technology [7]. All these designs can only handle short reads.

3 Pre-alignment Filtering for Long Reads

The proposed mechanisms for pre-alignment filtering in Section 2.2 are not directly applicable to long reads. Applying the filtering algorithm without adjusting the bin and token sizes for long reads would yield near-zero filtering rates due to high-magnitude CountBuffer values. Conceptually, increasing both the token and bin sizes might allow direct long read filtering, but that requires considerable changes to the hardware; *e.g.*, the bit vectors' and CountBuffer's memory footprint increase exponentially with token size, making the design impractical. For instance, increasing the token size by one increases the memory footprint of the bit vectors by $4\times$.

To filter long reads without significant overhead, we keep the bin and token sizes unchanged and instead split the long reads into *chunks* with sizes comparable to the reference bins. Each chunk is compared with successive bins in the reference genome, and the similarity score of individual chunks is accumulated to compute the overall similarity score. For instance, in Fig. 4, the long read q_i is divided into $N = 4$ chunks: $\mathtt{chk}_0$, $\mathtt{chk}_1$, $\mathtt{chk}_2$, and $\mathtt{chk}_3$. In order to find the mapping similarity score of q_i at any location in the reference genome, referred to as a *logical* bin, ρ_j, all chunks of q_i having separate count buffers are compared to successive bins in R starting at the location r_j. For instance, if we want to compute the similarity of q_i with ρ_0 at location r_0, $\mathtt{chk}_0$ is compared to r_0, $\mathtt{chk}_1$ to r_1, $\mathtt{chk}_2$ to r_2, and $\mathtt{chk}_3$ to r_3 and the similarity score of the comparisons is accumulated: $\sum_{i=0}^{i=N-1} \sum_{j \in \Theta(chk_i)} \text{CountBuffer}^{chk_i}(j) * r^v_{k+i}[f(j)]$ where k is the starting position in the reference genome (0 in this case).

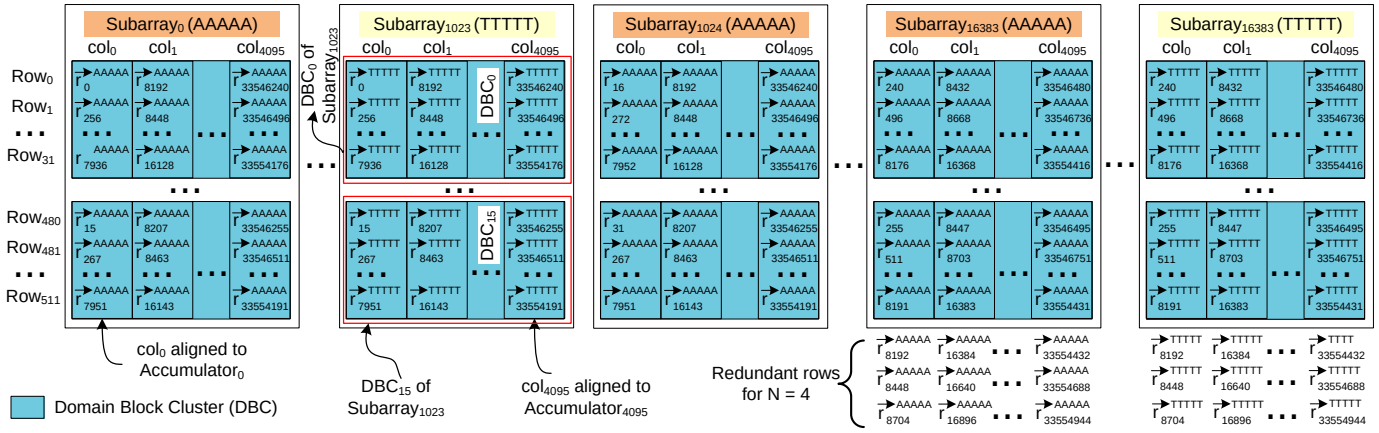

Fig. 3: Our optimized data placement for long reads, with backward compatibility for short reads.

Intuitively, the proposed *split-and-compare* mechanism for long reads should not affect the filtering accuracy as we retain the small token window of short reads. Nonetheless, we validate the correctness of our filtering algorithm by comparing its output with the BWA-mem and minimap2 aligners. We ensure that the bin positions filtered out are not contained in the output of the two aligners.

3.1 CNM Architecture for Short and Long Reads

FIRM [7] only handles short reads. In order to use the FIRM accelerator for long reads, it requires data duplication and column shifting. To explain it further, let us assume we want to compare the first 4 k logical bins (*i.e.*, ρ_0 to ρ_{4095}) in R with a given long read q_i having $N = 4$. For the comparison to work, the last $N - 1 = 3$ logical bins (*i.e.*, ρ_{4093}, ρ_{4094}, and ρ_{4095}) requires three redundant columns for r_{4096}, r_{4097}, and r_{4098} in each subarray. Second, in order to use the same accumulator for all N chunks, this data layout requires shifting each fetched row for alignment prior to summation, as in the accelerator design a single accumulator to use to compute the similarity score of one entire reference bin. The shifting problem can be avoided by using N accumulators per long read. However, this requires considerable changes and hardware overhead to the accelerator design to sum up all the partial results of the N accumulators. This also incurs energy overhead and reduces the parallelism degree from 4096 to $4096/N$.

Alternatively, placing successive bins (belonging to the same logical bin) in adjacent rows also solves the accumulator alignment problem. However, in RTM-based systems, it may result in a notable increase in RTM shifts. In the following, we introduce a novel data placement mechanism that resolves the accumulator alignment problem while reducing the number of RTM shifts. In the following, we propose a novel data placement mechanism that solves the accumulator alignment problem while reducing the number of RTM shifts.

3.1.1 Optimized Data Placement for Long and Short Reads

Fig. 3 shows our novel data layout that supports both long and short reads. Note that the subarray layout (e.g., Subaary$_0$) is similar to Fig. 1, i.e., each subarray has 16 DBCs, each DBC has 4096 nanowires (number of columns) and each nanowire has 32 bits (e.g., DBC$_0$ contains Row$_{0-31}$). In the proposed mapping, presence bits of the reference bins are stored in a subarray-interleaved fashion, as indicated by the subarray labels (e.g., Subarray$_0$ (AAAAA), meaning this subarray exclusively stores presence bits for token AAAAA). This interleaving reduces the number of active subarrays and minimizes within-subarray movements (RTM shifts), as subarrays dedicated to tokens that are not present in the query read are never accessed. Presence bits of successive bins (e.g., r_{0-3}) in a logic bin (ρ_0) are placed in the same column (col$_0$) and the same index of adjacent DBCs (DBC$_{0-3}$). The row and column offsets (see Fig. 1) in our layout are parameterized so that they can work with different memory sizes and configurations. For instance, for the memory configuration in Fig. 3, column offset is determined as $S \times W/1024 = 8192$, where S is the number of subarrays and W is the rows per subarray. The row offset is $o = D \times S/1024 = 256$, where D is DBCs per subarray. This layout does not require column shifting as all bins belonging to logical bin ρ_j are already aligned to the same accumulator.

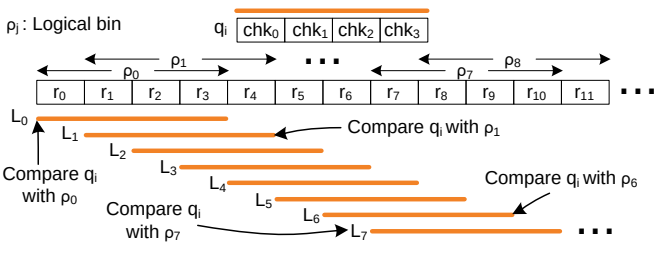

Fig. 4: Overview of the pre-filtering for long reads

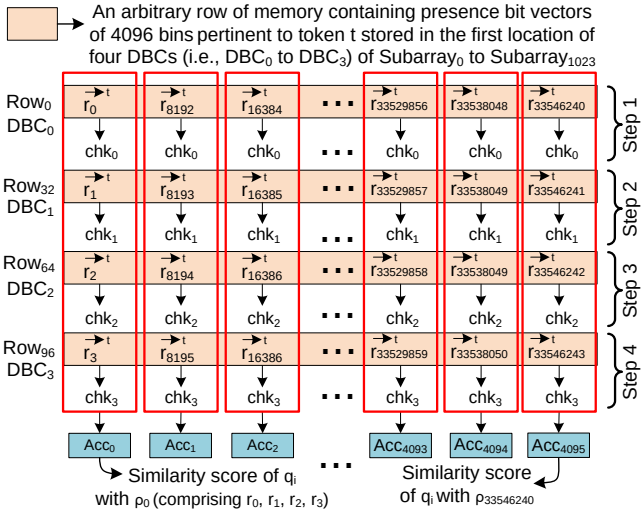

Fig. 5: An example showing how the novel data placement solves the accumulator misalignment problem for long reads.

To demonstrate how this layout solves the accumulator alignment problem, consider the example in Fig. 5 where all bins of ρ_0 (*i.e.*, r_0, r_1, r_2, and r_3) are aligned to Acc$_0$. Similarly all bins pertinent to $\rho_{33546240}$ are aligned to Acc$_{4095}$. Compared to FIRM's layout where $(N-1) \times S$ redundant columns were needed for long reads, which accumulates to a considerable 19.3% storage overhead for a subarray with $W = 512$, our

TABLE 1: Energy and latency values [13], [14], [15]

Operation	Value
Activation and precharge: DRAM/RTM	1964/1087 [pJ]
DRAM energy: Access / IO	1.25/0.40 [pJ]/bit
RTM energy: Read / shift	0.76/0.23 [pJ]/bit
Background power: DRAM/RTM	410/212 [mW]
RTM shift latency	1.87 (2 cycles)
t_{RAS}-t_{RCD}-t_{RP}-t_{CAS}-t_{WR} (DRAM)	20-8-8-8-8 [cycles]
t_{RAS}-t_{RCD}-t_{RS}-t_{CAS}-t_{WR} (RTM)	9-4-2S-4-4 [cycles]
Accel. power per bit (dynamic/leakage)	$1.77{\cdot}10^3$ / 11.16 [mW]
Preproc. unit per bit (dynamic/leakage)	11.6 / 1.13 [mW]

new data placement for long reads requires $(N-1) \times 1024$ redundant rows (required for the last 1024 subarrays) for the entire memory, which is an overhead of 1.2% for a 8 GB memory. Note that this data layout also supports short reads and produces comparable results to the layout in FIRM [7].

4 Evaluation

For evaluation, we use four paired-end input short read genomes of size 100, same as in [7], and four long read genomes [12] of size 10K (L1-10kb, L2-10kb) and 15K (L1-15kb, L2-15kb). For the sake of comparison to the GRIM and ALPHA filters, we assume bin and token sizes of 100 and 5, respectively. We model an 8 GB memory having 64 banks and 512 subarrays per bank using the cycle-accurate RTM simulator RTSIM [16] The energy and latency numbers of the logical components and the memory subsystems shown in Table 1 are estimated using McPat [14], DRAMSpec [13], DESTINY [15], and Cadence RTL-Compiler Synthesis for the custom components on the logic layer, all presuming 32nm fabrication technology.

We compare the pre-alignment filtering runtime and energy consumption of our design with the state-of-the-art *GRIM [5], ALPHA [9] and FIRM [7]* designs. We also explicitly compare the naive FIRM layout (*CNM-LC*) with our shift and performance aware layout (*CNM-LI*) and report the energy and performance results. All results include the query read preprocessing, *i.e.*, populating the count buffers, loading the metadata of R and the data transfers between the memory and the logic layers.

4.1 Results and discussions

The performance and energy efficiency of the proposed RTM-based CNM system largely depends on access parallelism and shift cost. Naively extending the short reads' optimized layout proposed in FIRM for long reads, noted as CNM-LC, leads to performance and energy inefficiencies. As discussed in Section 3.1.1, our CNM-LI avoids unnecessary shifts compared to CNM-LC data placement, which reduces the shifting overhead in RTM by 11.8× for long reads. This shift reduction is because our CNM-LI design requires a single shift per access. As a result, CNM-LI reduces the execution time on average by 1.32× compared to CNM-LC (see Figure 6). It also reduces the energy consumption by 50% compared to the CNM-LC (see Figure 7).

For short reads, our proposed data mapping produces comparable results to FIRM [7] while outperforming GRIM and ALPHA by a factor of 3.6× and 3.4× respectively. Moreover, it reduces energy consumption by 2.5× and 2.3× compared to GRIM and ALPHA, respectively (see Figure 7). The energy results analysis shows that the background and the refresh energy account for a significant portion of the total energy in the DRAM-based designs, GRIM and ALPHA.

5 Conclusions

In this paper, we present an RTM based CNM system for energy-efficient pre-alignment filtering. We propose a novel data mapping that significantly reduces the data duplication and RTM shift operation overheads and maximizes parallelism.

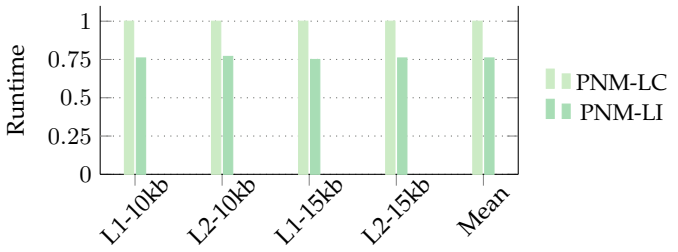

Fig. 6: Runtime comparison of CNM-LI and CNM-LC.

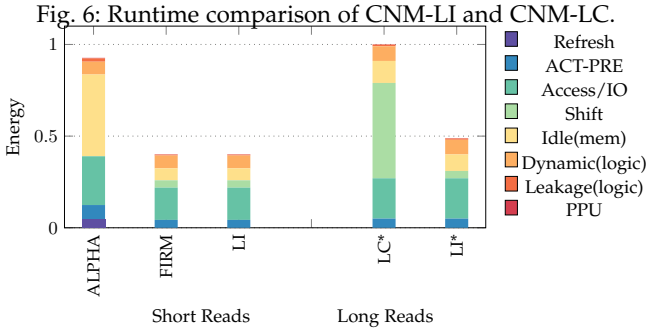

Fig. 7: Short and long reads energy comparison.

Our exploration of the design space and experimental evaluation show that our proposed layouts support both long and short reads. For short reads, our novel mapping produces comparable results to FIRM while for long reads, it reduces the execution time and energy consumption by 1.32× and 50%, compared to the data mapping presented in FIRM.

References

[1] R. C. Edgar et al., "Petabase-scale sequence alignment catalyses viral discovery," *Nature*, vol. 602, no. 7895, pp. 142–147, 2022.
[2] T. F. Smith and M. S. Waterman, "Identification of Common Molecular Sub sequences," *Journal of Molecular Biology*, vol. 147, no. 1, pp. 195–197, March 1981.
[3] M. Vasimuddin et al., "Efficient architecture-aware acceleration of bwa-mem for multicore systems," *IPDPS*, pp. 314–324, 2019.
[4] H. Li, "Minimap2: pairwise alignment for nucleotide sequences," *Bioinformatics*, vol. 34, no. 18, pp. 3094–3100, 05 2018.
[5] J. Kim et al., "GRIM-Filter: Fast Seed Location Filtering in DNA Read Mapping using Processing-in-memory Technologies," *BMC Genomics*, vol. 19, no. 2, 2018.
[6] Y. Turakhia et al., "Darwin: A hardware-acceleration framework for genomic sequence alignment," *bioRxiv*, 2017.
[7] F. Hameed et al., "Dna pre-alignment filter using processing near racetrack memory," *IEEE CAL*, no. 02, pp. 53–56, jul 2022.
[8] M. Alser et al., "SneakySnake: a fast and accurate universal genome pre-alignment filter for CPUs, GPUs and FPGAs," *Bioinformatics*, vol. 36, no. 22-23, pp. 5282–5290, 12 2020.
[9] F. Hameed et al., "ALPHA: A Novel Algorithm-Hardware Co-design for Accelerating DNA Seed Location Filtering," *IEEE Transactions on Emerging Topics in Computing*, pp. 1–1, 2021.
[10] G. Logsdon et al., "Long-read human genome sequencing and its applications," *Nature Rev. Genet.*, vol. 21, no. 10, pp. 597–614, 2020.
[11] R. Bläsing et al., "Magnetic racetrack memory: From physics to the cusp of applications within a decade," *Proceedings of the IEEE*, vol. 108, no. 8, pp. 1303–1321, 2020.
[12] "Genome in a bottle data indexes," https://github.com/genome-in-a-bottle/giab_data_indexes/tree/master/AshkenazimTrio, accessed: 2023-08-09.
[13] O. Naji et al., "A High-level DRAM Timing, Power and Area Exploration Tool," in *SAMOS*, July 2015, pp. 149–156.
[14] L. Sheng et al, "McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures," in *Int. Symp. on Microarch.*, 2009, pp. 469–480.
[15] S. Mittal et al., "Destiny: A comprehensive tool with 3d and multi-level cell memory modeling capability," *Jour. of Low Power Electronics and Applications*, vol. 7, no. 3, 2017.
[16] A. A. Khan et al., "RTSim: A Cycle-Accurate Simulator for Racetrack Memories," *IEEE Computer Architecture Letters*, vol. 18, no. 1, pp. 43–46, Jan 2019.