FindeR: Accelerating FM-Index-based Exact Pattern Matching in Genomic Sequences through ReRAM technology

Farzaneh Zokaee‡
Indiana University Bloomington
fzokaee@iu.edu

Mingzhe Zhang
ICT, CAS, China
zhangmingzhe@ict.ac.cn

Lei Jiang‡
Indiana University Bloomington
jiang60@iu.edu

***Abstract*—Genomics is the critical key to enabling precision medicine, ensuring global food security and enforcing wildlife conservation. The massive genomic data produced by various genome sequencing technologies presents a significant challenge for genome analysis. Because of errors from sequencing machines and genetic variations, approximate pattern matching (APM) is a must for practical genome analysis. Recent work proposes FPGA, ASIC and even process-in-memory-based accelerators to boost the APM throughput by accelerating dynamic-programming-based algorithms (e.g., Smith-Waterman). However, existing accelerators lack the efficient hardware acceleration for the exact pattern matching (EPM) that is an even more critical and essential function widely used in almost every step of genome analysis including assembly, alignment, annotation and compression.**

State-of-the-art genome analysis adopts the FM-Index that augments the space-efficient BWT with additional data structures permitting fast EPM operations. But the FM-Index is notorious for poor spatial locality and massive random memory accesses. In this paper, we propose a ReRAM-based process-in-memory architecture, FindeR, to enhance the FM-Index EPM search throughput in genomic sequences. We build a reliable and energy-efficient Hamming distance unit to accelerate the computing kernel of FM-Index search using commodity ReRAM chips without introducing extra CMOS logic. We further architect a full-fledged FM-Index search pipeline and improve its search throughput by lightweight scheduling on the NVDIMM. We also create a system library for programmers to invoke FindeR to perform EPMs in genome analysis. Compared to state-of-the-art accelerators, FindeR improves the FM-Index search throughput by $83\% \sim 30K\times$ and throughput per Watt by $3.5\times \sim 42.5K\times$.

***Index Terms*—short DNA alignment, ReRAM**

I. Introduction

High throughput sequencing technologies (i.e., Illumina [1], PacBio SMRT [2] and Oxford Nanopore [3]) have revolutionized biological sciences, since they can sequence an entire human genome within a single day. The explosion of genomic data has been the cornerstone in enabling the understanding of complex human diseases [4], ensuring global food security [5] and enforcing wildlife conservation [6]. Several government projects [7] around the world have been launched to deploy whole genome sequencing in clinical practice and public health. Genome analysis will become one of the standard practices of newborn screening over the next decade. The *speed* of genome analysis of big genomic data is a matter of life and death.

However, it is *challenging* to efficiently store, process and analyze the huge amounts of genomic data generated by sequencing machines that translate organic nucleotides to digital symbols. The genome sequencing machines are improving faster than Moore's Law [8]. For instance, a recent Illumina NovaSeq machine [9] produces nearly 750GB of data per day. Oxford Nanopore even creates USB-drive-style devices to sequence organisms in the wild [3]. It is projected that by 2025 the total amount of genomic data on earth will exceed the data capacity of YouTube and Twitter [10]. As a result, the exponential genomic data growth significantly increases pressure on hardware platforms for genome analysis. Analyzing a single genome may take hundreds of CPU hours [11], [12] on high-end servers. Application-specific acceleration for genome analysis has become essential.

Critical steps in genome analysis such as *assembly* and *alignment* involve both exact and approximate pattern matching [8], [13], because of their *seed-and-extend* paradigm [14]–[18]. Parts of a genome fragment (called seeds) are mapped to their matched positions in other fragments (assembly) or long references (alignment) by *exact pattern matching* (EPM) during seeding. Seed extension pieces together a larger sequence with seed mappings and edit distance errors, i.e., insertions, deletions (indels) and substitutions, by *approximate pattern matching* (APM). All genome fragments have to go through the seeding stage, but only the fragments containing at least one exactly matched seed are actually processed in the seed extension phase. So EPM operations during seeding cost even larger amounts of CPU time than APM operations during seed extension in state-of-the-art genome analysis software [19], [20]. Recent accelerators heavily optimize APM in seed extension by accelerating dynamic programming-based algorithms [11] (e.g., Smith-Waterman algorithm [21]–[23]) or Universal Levenshtein Automata [12], because APM handles edit distance errors and thus has higher time complexity. In contrast, for exact pattern matching (EPM) during seeding, these accelerators [11], [12], [21]–[25] use a simple hash table to find positions exactly matching a seed (hit) in a sequence. However, the hash-table-based technique is inefficient to implement EPM during seeding. For high sensitivity and precision, a huge number of seeds have to be used to produce large amounts of candidate positions with seed hits. The seeds and candidate positions may occupy $80GB$ [11] DRAM. Furthermore, false positive candidate positions generated by the hash-table-based technique slow down the seed extension.

The Ferragina-Manzini Index (FM-Index) [26] is adopted by state-of-the-art genome analysis software such as BWA-

‡This work was supported in part by NSF CCF-1909509.

978-1-7281-3613-4/19/$31.00 ©2019 IEEE
DOI 10.1109/PACT.2019.00030

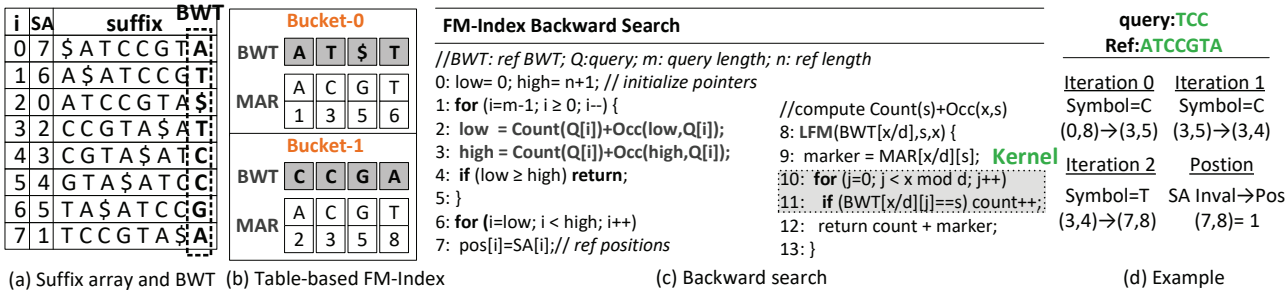


Fig. 1: The FM-Index overview: (a) SA and BWT; (b) FM-Index; (c) Backward search; (d) An example.

MEM [17] and SOAP [27] to build super-maximal exact matches (SMEMs) during seeding, since it augments the space-efficient Burrows-Wheeler transform [28] with accessory data structures that permit fast EPM. SMEMs generated by the FM-Index guarantee each seed does not overlap other seeds and has the maximal length that cannot be further extended. Compared to the hash-table-based technique, the FM-Index reduces not only the number of errors in final genome mappings but also the seed extension duration [29].

Besides genome assembly and alignment, FM-Index is also widely used in other time-consuming steps of genome analysis such as genome annotation [30] and compression [31], [32] for EPM. Recent work presents FPGA [19], [20], [32], [33] and ASIC [34]-based accelerators to process FM-Index searches. However, the FM-Index is notorious for its massive random memory accesses [35], [36], so it hits the *memory wall* of the *von Neumann* architecture. Existing FM-Index accelerators are fundamentally limited by memory bandwidth. In this paper, we propose a ReRAM-based process-in-memory architecture, FindeR, to accelerate FM-Index-based exact pattern matching (EPM) in genomic sequences. Our contributions are summarized as follows.

- **A ReRAM-based Hamming distance unit** – We transform a symbol-counting operation in the FM-Index search kernel to a Hamming distance calculation. We then propose a ReRAM-based Hamming distance unit (RHU) and a lookup-table-based adder to accelerate FM-Index searches in commodity ReRAM chips without extra CMOS gates.
- **Reliability, power and density** – We present architectural techniques to provide RHUs long lifetime and high current accumulation accuracy. To balance the trade-off between area, power and energy, we comprehensively explore the design space of FM-Index by tuning its bucket size.
- **A full-fledged pipeline with system support** – We further architect a full-fledged FM-Index pipeline and improve the search throughput by lightweight scheduling. Finally, we design a system-level interface for genomics developers to run FindeR for EPM.
- **EPM throughput per Watt** – We evaluate and compare FindeR to prior hardware platforms in various genome analysis applications such as genome assembly, alignment, annotation and compression. The results show that our PIM improves the FM-Index search throughput by $83\% \sim 30K\times$ and throughput per Watt by $3.5\times \sim 42.5K\times$ over the state-of-the-art accelerators.

II. Background and Motivation

A. Exact Pattern Matching in Genome Analysis

1) Genome Analysis

Genome analysis [13] mainly includes four steps: *sequencing*, *assembly*, *alignment* and *annotation*. In genome sequencing, sequencing machines translate organic nucleotide fragments to digital DNA sequences (called *reads*) comprising A, C, G and T. Assembly constructs larger DNA sequences by merging different reads, while alignment decides the precise order of nucleotides by aligning short reads to a long reference genome. Finally, annotation attaches biological information to long sequences.

2) Exact Pattern Matching

Genomic data have two significant sources of errors: sequencing machine errors and true variations [37]. The sequencing error rate of various sequencing machines is $0.2\% \sim 30\%$ [37], while the overall variation of the human population has been estimated as 0.1% [8]. So practical genome analysis has to support approximate pattern matching (APM) to accommodate errors. The APMs are often implemented by the Smith-Waterman (SW) algorithm [21]–[23] that has high time complexity and is impractically slow. To avoid the prohibitive SW overhead, the seed-and-extension paradigm is adopted in genome assembly [11], [16], [18] and alignment [8], [17]. The seeding stage heavily depends on Exact Pattern Matching (EPM) to find parts of a read that exactly match at least one position in another read or the reference. Compared to an APM, an EPM is more elementary, since an APM can be broken into multiple EPMs [8]. The FM-Index [30], [31], [33], [38] is one of the most computationally and memory efficient solutions to performing EPM.

B. FM-Index

1) Data Structure

The Ferragina-Manzini Index (FM-Index) [26] is built upon the Burrows-Wheeler transform (BWT) [28], a permutation of a symbol sequence generated from its suffix array (SA). The SA of a reference genome $\mathcal{G}$ is a lexicographically sorted array of the suffixes of $\mathcal{G}$, where each suffix is denoted by its position in $\mathcal{G}$. \$ indicates the end of a genome, so it is in the lexicographically smallest position. Each position of the BWT is calculated by

$$BWT[i] = \mathcal{G}[(SA[i] - 1) \bmod |\mathcal{G}|], \quad (1)$$

```
kMismatchSearch(Q, k, i, low, high)
//BWT: ref BWT; Q: the short read; k: the mismatch #; i: the symbol index of Q;
0: if (low ≥ high) return 0;
1: if (i = 0) return [low, high];
2: foreach ch in {A,C,G,T} {
3:   low = LF-M(BWT[low/d] , ch, low);
4:   high = LF-M(BWT[high/d], ch, high);
5:   if (Q[i] != ch) k = k - 1;
6:   if (i ≥ 0) kMismatchSearch(Q, k, i-1, low, high)
7: }
```

Fig. 2: The FM-Index-based k-mismatch search.

where $|\mathcal{G}|$ is the length of $\mathcal{G}$. Figure 1(a) shows the SA of a reference sequence $\mathcal{G} = ATCCGTA\$$ and its $BWT(\mathcal{G}) = AT\$TCCGA$. The FM-Index implements EPM searches through two functions $Count(s)$ and $Occ(s, i)$ performed on the BWT. $Count(s)$ computes the number of symbols in the BWT that are lexicographically smaller than the symbol s, e.g., $Count(T) = 6$. $Occ(s, i)$ returns the number of symbol s in the BWT from positions 0 to $i - 1$, e.g., $Occ(C, 5) = 1$. The values of $Count(s)$ and $Occ(s, i)$ can be pre-calculated and stored, but the storage overhead is significant. To keep the FM-Index size in check, the Occ values are sampled into buckets of width d shown in Figure 1(b) ($d = 4$). The Occ values are stored each d positions as markers (MARs) to reduce the storage overhead by a factor of d. The omitted Occ values can be reconstructed by summing the previous marker and the number of symbol s from the remaining positions in the BWT bucket. We call this data structure *table*-based FM-Index. To simplify search operations, the $Count$ values for each symbol are added to the corresponding markers. The markers and the BWT buckets are interleaved to build the FM-Index. The FM-Index size (F) can be calculated by Equation 2, where $|\Sigma|$ is the alphabet size. The first item in the equation indicates the Occ table size, while the second represents the BWT size. For a human genome ($|\mathcal{G}| = 3G$, $|\Sigma| = 4$ and $d = 128$), the FM-Index costs 1.5GB.

$$F = \frac{4 \cdot |\mathcal{G}| \cdot |\Sigma|}{d} + \frac{|\mathcal{G}| \cdot \lceil log_2(|\Sigma| + 1) \rceil}{8} \quad bytes \qquad (2)$$

2) Backward Search

EPM is achieved by the FM-Index backward search, whose algorithm can be viewed in Figure 1(c). The SA interval $(low, high)$ covers a range of indices in the SA where the suffixes have the same prefix. The pointer low locates the index in the SA where the pattern is first found as a prefix, while the pointer $high$ provides the index after the one where the pattern is last found. At first, low and $high$ are initialized to the minimum and maximum indices of the Occ array respectively. And then, iterating from the last symbol in the read to the first, the SA interval is updated using the Last-First Mapping (*LFM*) function where low is calculated:

$$Count(Q[i]) + Occ(Q[i], low). \qquad (3)$$

$high$ can be computed in the same way. The function of *LFM* is shown from line 8 to 13 in Figure 1(c), where the computations of low and $high$ are **random pointer chasing having poor spatial locality**. Finally, the SA interval gives the range of indices in the SA where the suffixes have the target read as a prefix. These indices are converted to reference genome positions using the SA. Figure 1(d) illustrates an example of searching TCC in the reference $\mathcal{G} = ATCCGTA$. Before a search happens, $(low, high)$ is initialized to $(0, 8)$. In Iteration 0, the last symbol C is processed by *LFM* where $(low, high)$ is updated to $(3, 5)$. After three iterations, $(low, high)$ eventually equals $(7, 8)$. By looking up $SA[7]$ in Figure 1(a), we find that the read TCC in reference $\mathcal{G} = ATCCGTA$ starts from $\mathcal{G}[1]$. Once the FM-Index is built, huge volumes of backward searches are invoked by almost every step of genome analysis.

3) The FM-Index for k-Mismatch Search

Compared to an APM, an EPM is more elementary, since an APM can be transformed into multiple EPMs to handle indels and substitutions [8]. However, the FM-Index-based k-error search that is able to handle both indels and substitutions exponentially increases the number of iterations in the *LFM* function. So the seed extension phase of long read alignment whose major error type is indel uses only the SW algorithm [11]. On the contrary, the FM-Index-based k-mismatch search that can accommodate only substitutions moderately increases the iteration number of the *LFM* function. Considering that 98.9% of short read sequencing errors [37] are mismatches, the state-of-the-art aligners such as BWA-MEM [17] and SOAP [27] implement the **APMs** to process mismatches in the seed extension phase of short read alignment by **k-mismatch FM-Index searches** [8].

The algorithm of k-mismatch FM-Index search is shown in Figure 2. Its main difference from the normal FM-index backward search (Figure 1(c)) lies in line 2, where in addition to the current symbol of the read, all possible substitutions are tested until the substitution number exceeds k. To search in both directions, the FM-Index is generated for both the BWT and its reverse [39]. The bi-directional FM-Index searches from the middle of a read, uses the forward search for the first half, and adopts the backward search for the second half. The iteration number of k-mismatch FM-Index is decided by both the read length and the mismatch positions.

4) FM-Index Applications

The FM-Index has been widely adopted for EPM in various steps of genome analysis:

- **SMEM Seeding**. The state-of-the-art genome aligners, e.g., BWA-MEM [17] and SOAP [27], create super-maximal exact matches (SMEMs) during seeding by FM-Index. A SMEM is an exact match in a read that cannot be extended further in either direction or contained in any other SMEM. Compared to hash-table-based seeding, SMEM seeding reduces not only the number of errors in genome mappings but also seed extension duration [29]. The detailed algorithm for SMEM construction can be found in [18]. The construction of SMEM first extends the seed rightward by FM-Index searches. The SA intervals of the hits sharing the same starting position are stored in an interval set. The construction of SMEM further leftward extends the hits found in rightward seed extensions by FM-Index searches

too. The FM-Index search has been identified as the most time-consuming operation for SMEM constructions [40].

- **Seed Extension for Short Reads**. The error rate of Illumina sequencing machines for short (100-bp) reads is only $\sim 0.2\%$ [37]. Rather than indels, the majority of the Illumina errors are substitutions [37] (mismatches). The overall variation in the human population is 0.1% [8]. 74% of 100-bp reads have no mismatch. 22.4% have 1 mismatch, while 3.3% have 2 mismatches. The state-of-the-art aligners [17], [27] use k-mismatch bi-directional FM-Index searches for short read seed extensions. 99.7% of short reads [33] are actually aligned by the 2-mismatch bi-directional FM-Index.
- **Genome Annotation**. A keyword comprising multiple symbols in Σ has a certain number of exact matches within a genome. Keyword counting is indispensable in annotation applications such as probe design, discovery of repeat elements, and mathematical modeling of genome evolution. The FM-Index search is the key operation for computing the keyword count [30].
- **Genome Compression**. The genomic big data explosion produced by emerging sequencing technologies poses a profound storage challenge. Biological sequences of the same species are highly similar and differ only by single nucleotide polymorphisms. The reference-based compression algorithm [31] records only the differences between a genome sequence and the reference, so it achieves higher compression ratios than general purpose compressors including gzip and bzip2 [32]. The FM-Index searches deciding the mapping to the reference account for 70% of the compression time [31].

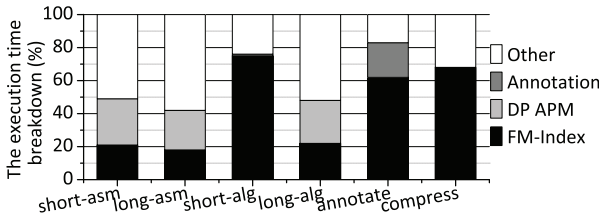


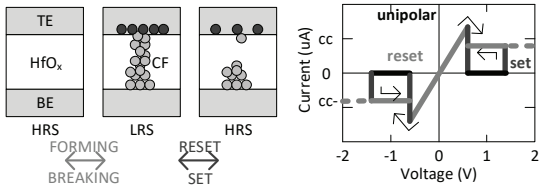
Fig. 3: Execution time breakdown in genome analysis.

Figure 3 shows the execution time breakdown of genomic applications. Our experimental methodology is elaborated in Section IV. On average, **FM-Index searches cost** $18\% \sim 75\%$ **of the total execution time in these applications**. Particularly, the FM-Index search is heavily invoked in short read alignment, since SMEM seeding and most seed extensions rely on it. In short read alignment, the FM-Index-based EPM consumes significantly more CPU time than that spent by the dynamic programming (DP) APM. APMs spend $24\% \sim 28\%$ of the total execution time in assembling short and long reads, and aligning long reads. But the other steps of genome analysis hardly use FM-Index searches.

C. ReRAM for Big Genomic Data

1) Processing Genomic Data in ReRAM

Because of its high density, long cell endurance, and low write power consumption, ReRAM [21], [24], [25] is deemed one of the most promising nonvolatile memory technologies to overcome the big genomic data challenge. Through the $\mathbf{4F^2}$ cell size, multi-level cells and cross-point array structure, a ReRAM-based main memory maintains scalable performance for genome analysis applications. However, recent work uses low-density and power-hungry ReRAM-based content address memories (CAMs) to perform hash-based short read seeding [24], [25] or dynamic-programming-based APMs [21] for short read seed extensions. Compared to APM, EPM consumes even more time in various genomic applications. However, **no prior work focuses on accelerating FM-Index-based EPM operations by ReRAM technology**.



(a) Cell. (b) Switching.

Fig. 4: The unipolar switching ReRAM.

2) Basics

A ReRAM cell storing data in a thin layer of metal-oxide (HfO_x) is sandwiched by a top electrode (TE) and a bottom electrode (BE), as shown in Figure 4(a). By injecting electrical pulses, it can be switched to a high resistance state (HRS) or a low resistance state (LRS). The transition from HRS to LRS is called *SET*, while the reverse is named *RESET*. To initiate the switching between HRS and LRS in a metal-oxide layer, a ReRAM cell needs a conductive filament (CF) created to connect the TE and BE by a high voltage *FORMING* operation. Without the CF, the cell cannot be RESET or SET, but stays only in HRS. A RESET yields a HRS cell by rupturing cracks on the CF, while a SET recovers the complete CF resulting in a LRS cell. Through a *BREAKING* operation ($\sim 100\mu s$), the CF in a cell can be eliminated. There are two methods to implement ReRAM writes [41]: *bipolar switching* and *unipolar switching*. ReRAM cells are connected by bit-lines (BLs) and word-lines (WLs) to form an array. We define the voltage difference generated by a high (low) WL voltage and a low (high) BL voltage as the positive (negative) voltage. Bipolar switching sets cells by positive voltages and resets cells by negative voltages. On the contrary, as Figure 4(b) exhibits, during unipolar switching, both SETs and RESETs are completed with either positive or negative voltages [42].

3) The Incompatibility of ReRAM and Logic Processes

The ReRAM fabrication process and the logic fabrication process are different and often incompatible [43]. It is difficult to fabricate circuits with a ReRAM process, or ReRAM with a logic process. The gates in a ReRAM process are fully optimized for density and low leakage power by sacrificing performance. The ReRAM process often has < 5 metal layers, while the logic process has > 12 metal layers. The circuits fabricated in a ReRAM process suffer from much higher interconnect overhead. On the contrary, building ReRAM cells in a logic process creates embedded ReRAM [44], which increases

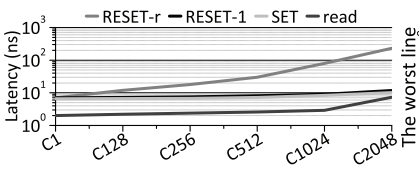


Fig. 5: The RHU latency.

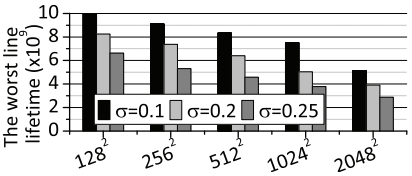


Fig. 6: The average RHU lifetime.

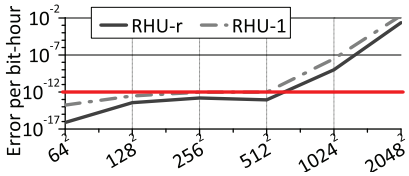


Fig. 7: The RHU accuracy.

the cell size to $\mathbf{44F^2}$. Therefore, combining ReRAMs and logic circuits results in low area efficiency and high power consumption. Our FindeR aims to process the kernel of FM-Index searches without CMOS logic circuits and by ReRAM arrays only.

III. FindeR

As Figure 1(c) exhibits, the bottleneck of an FM-Index backward search of an m-bp read lies in the *LFM* function calculating $Occ(s, i)$, since the LFM function is invoked by both low and $high$ for m times. The function of *LFM* counts the number of times the symbol s appears in a d-bp BWT bucket for the FM-Index. We transform a symbol counting operation to a Hamming distance (HD) calculation. Counting the symbol s in a d-bp BWT bucket can be computed by subtracting hd from d, where hd is the HD between the d-bp BWT bucket and a d-bp sequence where all symbols are s. We propose a reliable, fast and energy-efficient ReRAM-based HD unit (RHU) to accelerate HD computations. To avoid introducing extra CMOS logic gates, we present a ReRAM-based lookup table (LUT)-based adder to subtract hd from d inside a ReRAM chip. We further architect a full-fledged pipeline with lightweight scheduling and a system library to improve the FM-Index search throughput.

A. ReRAM-based Hamming Distance Unit

1) An RHU for the FM-Index

Figure 8 illustrates an RHU. In a 4×4 ReRAM array, only 4 main diagonal cells are used to calculate the HD between a 2-bp BWT bucket and a 2-bp read "ss", where s can be A, C, G and T. All the other cells are in HRS. But they cannot be SET or RESET, since they are not FORMed and thus have no filament. There are three steps to compute the HD value between the BWT bucket and the read, e.g., GA and TT. First, all diagonal cells in the array are initialized to HRS before an HD calculation. Second, we encode A, C, G and T by 00, 01, 10 and 11, and use $0V$ and $1.5V$ to indicate 0 and 1. $1.5V$ is the minimal voltage triggering unipolar SETs. But it is not high enough to start FORMING operations [41]. The corresponding voltage levels indicating the binary encoding of DNA symbols are applied on the WLs and BLs, respectively. For instance, the voltage corresponding to "1000" (GA) is applied on BLs and the voltage corresponding to "1111" (TT) is assigned to WLs. Each cell on the main diagonal line in the array remains in HRS if its WL and BL have the same voltage. The cell ❶ is in this case. Otherwise, the cell is switched to LRS. So the cells ❷∼❹ are switched to LRS. One DNA symbol mismatch generates one or two LRS cells. Third, the sensing voltage ($1V$) is applied to all BLs, while all WLs are grounded. A current limiting transistor connected to a pair of BLs passes at most only $1\times$ LRS sensing current even when both cells on two BLs are in LRS. The cells ❶ and ❷ supply $1\times$ LRS sensing current, while the cells ❸ and ❹ provide another $1\times$ LRS sensing current, although both of them are in LRS. The summed current representing the HD between GA and TT is measured and translated to a digital HD value (hd) by an analog-to-digital converter (ADC) shared by multiple RHUs. hd is 2 in this example. Instead of counting the symbol s, the HD indicates the number of symbols different from s in the BWT bucket. The number of symbol s in a d-bp BWT bucket is calculated by $d - hd$. Therefore, the number of T in GA is $2 - 2 = 0$. An RHU is a normal ReRAM array that can be accessed by both SAs and WDs in a chip.

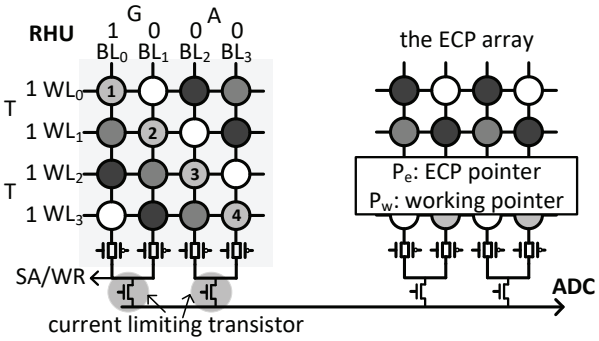


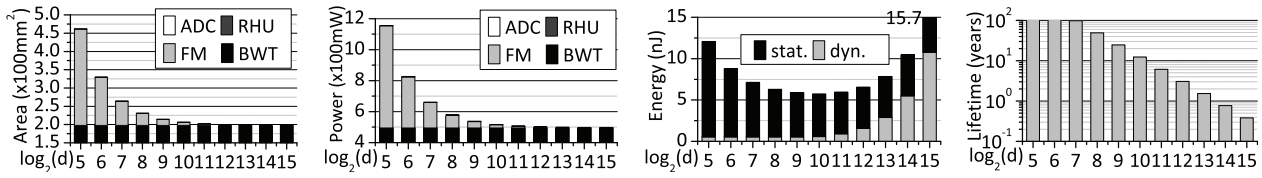
Fig. 8: An RHU in a ReRAM bank.

2) The Fabricated RHU Demonstration

The basic function of a unipolar switching RHU has been examined in [42]. However, the existing RHU demonstration relies on one line, but not an array, to perform HD calculations, since it cannot isolate the other cells in the array by skipping their FORMING operations. The existing RHU cannot be used to compute the HD between two genomic reads, since each symbol in a read is encoded by two bits. Without the current limiting transistor, an RHU will produce wrong HD values, since two LRS cells may be generated by a mismatch between two symbols. Moreover, an RHU fails within minutes when constantly computing HD values, due to the short ReRAM cell endurance. We propose wear-leveling and error-correcting schemes to provide RHUs sufficiently long lifetime. We shorten the initialization, HD calculation and popcount latencies of RHU and enhance the current accumulation accuracy of RHU during popcounts by tuning the RHU array size.

3) The RHU Latency

An RHU HD computation involves a RESET initialization, a SET and a read operation. The latencies of cells at different positions on a BL are shown in Figure 5. We adopted the ReRAM cell and array models from [45]. We define $Cell_1$



(a) Area. (b) Power. (c) Energy. (d) Lifetime.

Fig. 9: The design space exploration of RHU with varying BWT bucket widths (d). (**All Y-axes are** $log_2(d)$)

($C1$) as the cell closest to the sense amplifiers (SAs) and write drivers (WDs), while $Cell_{2048}$ ($C2048$) is the cell farthest from the WDs on a BL. The SET latency is insensitive to cell position on the BL and only slightly increases for cells far from the WDs. In contrast, as prior works [45]–[47] indicate, the RESET latency is exponentially prolonged by the voltage drop introduced by the BL resistance and sneak path current. The sneak path is heavily influenced by the data pattern in a ReRAM array [47]. If more cells on a BL or WL are in LRS, a larger sneak path current emerges on the BL or WL, and thus the RESET latency is longer. With a random data pattern (the one and zero ratio: 1-to-1), the cells far from the WDs on the BL suffer from large RESET latencies (*RESET-r*). However, the data pattern (*RESET-1*) where only main diagonal line cells are in LRS and all other cells are in HRS does not significantly increase the RESET latency even for cells far from the WDs, since for each pair of connected BL and WL there is only one LRS cell. A RESET on $Cell_{1024}$ on the BL takes only $9ns$ in an RHU. A long BL also slows down a read operation because of the larger RC delay. Since every HD calculation resets the RHU and sets the main diagonal line cells, the RHU can use a high voltage to accelerate the read operation without worrying that the high voltage may disturb cells. We adopted $1.0V$ as the read voltage during an HD calculation.

4) The RHU Endurance

Each ReRAM cell normally tolerates 10^{10} writes [41], but some weak cells last for only $10^5 \sim 10^6$ writes because of process variation. Since the RHU uses only the white main diagonal line in Figure 8, it fails within minutes when constantly computing HDs. To prolong the endurance of an RHU, we propose a wear leveling scheme to use a pair of other diagonal lines, e.g., two gray, light gray or dark gray lines, for an HD calculation in Figure 8. For each 100K HD calculations, we perform BREAKING operations to eliminate filaments in all cells of the current diagonal line pair, and conduct FORMING operations on another diagonal line pair. We use a pointer P_w to indicate which diagonal line pair is working in an RHU. P_w costs $log_2(w)$ bits, where w is the array width. To further improve the RHU lifetime, we adopted six Error Correcting Pointers (ECPs) [48] for the working diagonal line pair. If the white main diagonal line is working and its first cell fails (Figure 8), we use a pointer P_e to record the broken cell position in the diagonal line and two BLs in the ECP array to replace BL_0 and BL_1. With the help from error detecting units [48] in ReRAM chips, the RHU can reuse the same 6 P_es for all its diagonal line pairs. Six P_es occupy $6log_2(w)$ bits in a dedicated array. For instance, for a 1024×1024 array, we have 12 BLs as the BL replacement in the ECP array and 70 bits for P_w and P_es in a dedicated array. We modeled the cell endurance variation among 4GB cells by the *normal* distribution, where $\mu = 10^{10}$ and $\sigma = 0.1 \sim 0.25$ [48]. The RHU lifetime is decided by its weakest diagonal line endurance with the protection of 6 ECPs and our wear leveling. The average RHU lifetime with varying array sizes is shown in Figure 6. On average, a larger RHU array has less ECPs per cell, so its lifetime is shorter. Even with $\sigma = 0.25$, the average lifetime of a 2048×2048 (2048^2) RHU is 2.87×10^9 writes. We will show the endurance of the whole FindeR PIM in Section III-D.

5) The RHU Accuracy

Based on Kirchoff's Law, an RHU sums currents passing through the cells in a diagonal line pair as the popcount result. We need to study the accuracy of current accumulation, which is influenced by the cell resistance variation, the wire resistance and the random telegraph noise (RTN) [49]. We adopted the ReRAM cell and array models from [45]. The LRS and HRS resistance variations are modeled through the *normal* distribution, where for LRS $\mu = 2k\Omega$, HRS $\mu = 2M\Omega$ and $\sigma = 0.03$ [50]. We modeled the RTN by adding 2% and 3% of binary noises to the LRS and HRS resistances [49]. We generated 4GB cells to study the accuracy of current accumulation. We produced random data (the one and zero ratio: 1-to-1) as the normal data pattern (*RHU-r*) and used *all-1*s as the worst case data pattern (*RHU-1*) for the cells in each diagonal line pair. Figure 7 exhibits the RHU current accumulation accuracy. RHU counts the number of LRS cells in a diagonal line pair. Unlike deep learning applications, FM-Index searches for exact pattern matching in genome sequences are more vulnerable to errors resulting in wrong pattern matching. The RHU should have similar reliability to a CMOS-based counterpart with SRAM registers. So we used the SRAM soft error rate, 10^{-12} error per bit-hour [51], as an acceptable error rate for the current accumulation (popcount) of RHU. The RHU can guarantee such low error rate with the worst *all-1*s data pattern (*RHU-1*) when counting all cells along a diagonal line pair even in a 512×512 array (512^2). This is because all of the cells of the RHU are in HRS except the cells in the current diagonal line pair. The structure of two BLs sharing one current-limiting transistor also alleviates the negative impact of the cell resistance variation and RTN.

B. ReRAM Lookup Table-based Adder

To avoid adding extra CMOS logic into a ReRAM chip, we propose a ReRAM lookup table (LUT)-based adder to calculate the sum of a marker and an RHU output. Figure 10 illustrates a ReRAM-based 8-bit LUT-based adder with three inputs $A_7 \sim A_0$, $B_7 \sim B_0$ and C_{in}. Through $A_7 \sim A_0$,

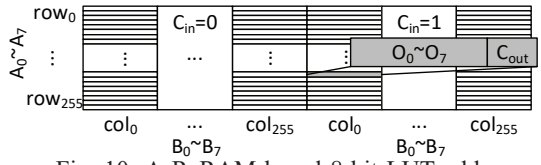


Fig. 10: A ReRAM-based 8-bit LUT adder.

the adder activates the corresponding WL. It uses $B_7 \sim B_0$ and C_{in} to select BLs to read the 9-bit result, including an 8-bit sum $O_7 \sim O_0$ and a 1-bit C_{cout}. The 8-bit LUT-based adder occupies 0.14MB cells. Because the markers of the FM-Index are 32-bit, we conduct 32-bit additions by four lookups on the 8-bit adder. We first send the 8 least significant bits (LSBs), $LSB_7 \sim LSB_0$, of the marker and the RHU output to the adder with $C_{in} = 0$. After the first lookup is done, the next 8 LSBs of two operands are assigned to the adder with $C_{in} = C_{cout'}$, where $C_{cout'}$ is the output of C_{cout} from the previous lookup. The lookups continue until all bits of two operands have been processed. To sum a marker and an RHU output hd, the FM-Index marker is modified to $marker + d$, where d is the FM-Index bucket width. And the LUT-based adder actually stores the values of $A_7 \sim A_0$ minus $B_7 \sim B_0$ to compute $marker + d - hd$.

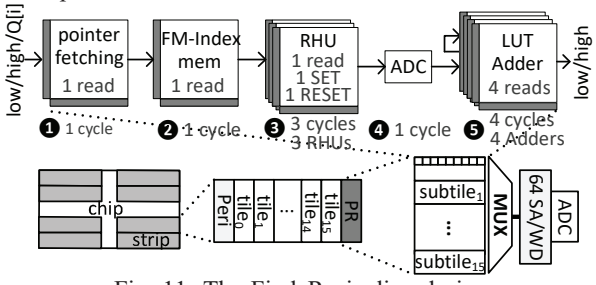


Fig. 11: The FindeR pipeline design.

C. Pipeline Design

1) Pipeline Details

FM-Index searches need to invoke the LPM function to update low and $high$. We propose a pipelined FindeR in Figure 11 to increase the throughput of the LPM function. The pipeline includes five stages: pointer fetching, FM-Index memory reading, RHU, ADC and adder. ❶ During the pointer fetching stage, the FM-Index searches read the working array pointer, the working diagonal line pointer (P_w), and ECPs (P_es) from their dedicated arrays. ❷ The FM-Index then fetches a d-symbol BWT bucket ($2d$-bit vector) with markers from FM-Index data arrays. ❸ The BWT (bit vector) and the read symbol are processed by an RHU specified by the pointers retrieved during ❶. The summed current of an RHU is sensed and held by a transimpedance amplifier (TIA). Besides a SET and a read, for each HD calculation, the RHU also needs to conduct a RESET initialization applying $V_{RESET} = 1V$ to all BLs and $0V$ to WLs without compliance current. So we require three RHUs to create an analog HD value each pipeline cycle. ❹ The current is converted to a digital value hd by a 128MHz 8-bit ADC [52]. ❺ The value of the $d-hd+marker$ for the FM-Index is calculated by looking up the adder. One addition consists of four lookups, each of which can be done within one pipeline cycle. So we integrate four adders into our pipeline to produce a result every cycle. We set one pipeline cycle as $10ns$, since as Section III-A explains, each 1024×1024 RHU read, SET, RESET and ADC conversion can be completed within $10ns$. In this way, the pipeline can be operated at $100MHz$. The function LFM of the FM-Index takes $90ns$ (i.e., pointer $10ns$, data $10ns$, RHU $20ns$, ADC $10ns$ and adder $40ns$) to calculate a low or $high$ by this pipeline design.

2) Enabling Strip-Level Parallelism

To process big genomic data, we assume a high density ReRAM-based main memory system [46], where eight ReRAM chips are interleaved to form eight banks through an NVDIMM. One FindeR pipeline is integrated into each bank, so that each pipeline can operate independently. In our ReRAM-based main memory, one bank can only serve one 64B read or write at one time. Each bank has eight strips, each of which is in a chip. Each strip is equipped with 64 SAs and 64 WDs as shown in Figure 11. To build a pipeline design inside each bank, each strip has to serve a smaller size access independently. We adopted the low overhead two-dimensional bank subdivision [53] to enable multiple simultaneous small size accesses in a bank. We add four 0.14MB arrays, each of which has 9 sense amplifiers and 9 write drivers, as the LUT adders in each bank. With the exception of adder arrays, FindeR requires only another 5 independent strips to run the pipeline in a bank.

D. FindeR PIM

1) Design Space Exploration

Figure 9 highlights the design space exploration of the RHU for the FM-Index with varying bucket widths ($log_2(d)$s). To achieve high memory density, we used $\mathbf{1024 \times 1024}$ arrays to build FindeR, since the SET, RESET and read operations in such large arrays can be completed within a pipeline cycle **10ns**. With increasing bucket size, the FM-Index storage is substantially reduced. The area and power for FM-Index, shown in Figure 9(a) and 9(b) decrease with a larger $log_2(d)$, since the FM-Index occupies less arrays. However, the LFM function has to count a symbol in a larger BWT bucket when d increases. The RHU has to SET more cells to complete an HD calculation within the one pipeline cycle. As Figure 9(d) exhibits, the lifetime of 16K 1024×1024 RHU arrays (**2GB**) decreases when d increases, because more writes occur in an RHU array during each HD calculation. With $log_2(d) = 10$ and large process variation ($\sigma = 0.25$), a FindeR consisting of 2GB RHU arrays can still stand for more than **10 years** even when constantly computing HD values. More concurrent SETs also substantially boost the RHU dynamic energy consumption in Figure 9(c). Considering that the RHU current accumulation accuracy is acceptable when counting $\leq$ 512 cells in a diagonal line, we conservatively selected $\mathbf{d = 128}$ to balance the area, power, endurance and current accumulation accuracy for the RHU.

2) Search Iteration Scheduling

A slave memory controller (SMC) [54], [55] is deployed on the NVDIMM to translate read/write requests from CPUs to

ReRAM device-specific commands (CMDs) that can be processed by banks. For the FM-Index, the SMC issues requests with read symbols to perform backward search iterations. The request addresses are calculated through low, $high$ and the bucket width d. After receiving the new low and $high$, the SMC schedules the next search iteration of the FM-Index as follows: when $low < high$, the low and $high$ will be returned to the on-chip master MC. A backward search is done. When $low > high$, the SMC creates requests, decodes their bank numbers by their address, and issues them into bank queues. When $low == high$, two requests for low and $high$ ask for the same BWT bucket. The SMC can coalesce these two requests. It issues two requests into the same bank queue and sets an indication bit in the low request, so that the bucket data can be kept in sense amplifiers of the bank. The $high$ request reads the data directly from SAs. To maintain the pipeline frequency, the coalescing does not reduce the $high$ request latency, but it minimizes the read energy. Based on our experiments, during short read alignment of human genome, 85% of symbols in a 100-bp read can coalesce their low and $high$ requests on average.

```
//FindeR c++ code
0: #include<FINDER>
1: int * low = null; int * high = null;
2: int * pos=null; const char * query = "TCC";
3: bwt.backward_search(query, low, high);
4: pos=(int *) malloc((high-low) * sizeof(int));
5: for (i=low; i < high; i++) pos[i]=SA[i];
```

```
//FindeR configuraiton
BWT_Path: /home/data/BWT;
FM_Path: /home/data/FM;
SA_Path: /home/data/SA;
RHU#: 8; Alphabet: ACGT; Bank#: 8;
Bucket_Width: 128;
Optimization: Power or Capacity;
```

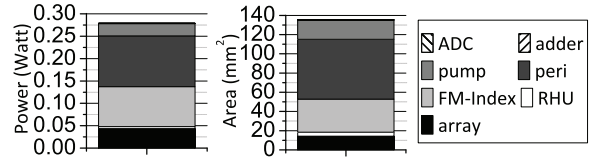
Fig. 12: The system support of FindeR.

3) System Support

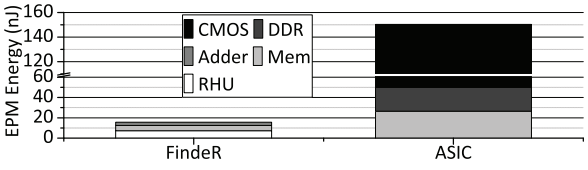
To run FindeR as a PIM accelerator in a computing system with processors, the FindeR integration stack includes three parts: configuration file, programming support and run-time library. Figure 12 describes a library configuration example and a code snippet of FindeR. In the *configuration* stage, programmers can easily specify the basic FindeR parameters, e.g., the BWT and FM-Index files, alphabet, FM-Index bucket width, bank number and RHU number, in the configuration file. We assume the BWT construction of the reference genomes and read pools are done in the cloud [56], [57], so that we can perform trillions of backward searches on them during all steps of genome analysis. At the beginning of compiling, the files of the BWT and FM-Index are copied into ReRAM chips and the other parameters are written into the SMC on the NVDIMM. The *programmers* can also enable FM-Index optimizations for FindeR in the configuration file by specifying optimization goals such as power or capacity. Based on the existing parameters, e.g., bank number, RHU number and alphabet, the FindeR library calculates the FM-Index bucket width to achieve the optimization goal. With the FindeR feedback and hint, programmers may re-compute the FM-Index for existing BWTs to attain smaller power and memory capacity locally or in the cloud. During programming, FindeR provides APIs for programmers to allow fast FM-Index-based EPM operations. When the compiled code *execute*s, FindeR accelerates FM-Index backward searches in ReRAMs and returns low and $high$ pointers to the CPU. To retrieve EPMs, the CPU fetches the SAs allocated in idle banks by low and $high$ pointers.

4) Hardware Overhead

We modeled the power, energy, latency and area of FindeR by NVSim [58] calibrated by the latest NVDIMM ReRAM power model [59]. FindeR consists of 8 banks, each of which supports a 64B access and is interleaved across 8 chips. When the bucket width is $d = 128$, FindeR stores the bi-directional FM-Index with 2.6GB ReRAM arrays in each bank. As Figure 13(a) shows, one 4GB ReRAM bank consumes $135mm^2$ area and $0.279W$ power at $32nm$. It spends $7.1nJ$ energy during each pipeline cycle. We assumed the FindeR NVDIMM has 8 4GB ReRAM banks (32GB) in the case that programmers may change d. Programmers can enable multiple banks to process FM-Index searches simultaneously. The idle arrays in each bank can be used for ECPs and main memory [60]. The bank subdivision [53] enabling independent strip accesses increases the ReRAM chip area by 5% and the power consumption by 3.2%. Four 8-bit LUT adders in each bank cost 0.56MB cells and require 36 SAs. An ADC is an essential component in ReRAM chips [50] for reference cell calibrations. We adopted an 8-bit ADC [52] in each bank and lowered its frequency to 128M Sample/sec to save power. We synthesized the ADC and NVDIMM scheduling logic by the Cadence design compiler with 32nm PTM process technology. The ADC costs $0.0012mm^2$ and $0.2mW$, while the scheduling logic consumes $0.00014mm^2$ and $0.001mW$. Unlike prior ReRAM-based convolutional neural network accelerators [61], [62], **the overhead of FindeR ADC power and area is trivial**, since it only handles binary HD computations but not floating point arithmetic. We show the energy consumption of an EPM in FindeR in Figure 13(b). Compared to an ASIC design [34] with 1.3GB DDR4 DRAM, FindeR reduces the energy per EPM by 89.6%. The DRAM has to open a 2KB row in each read, so even the energy consumed by DRAM arrays in an ASIC-based EPM is larger than the energy of an entire FindeR-based EPM.



(a) Area and Power.



(b) Energy.

Fig. 13: The hardware overhead of FindeR.

IV. Experimental Methodology

A. Simulation and evaluation

We modified the NVM timing simulator NVMain [63] to model the micro-architecture of FindeR. We implemented all

pipeline details including pointer and FM-Index arrays, RHUs, ADCs and LUT adders. To estimate the **throughput** of various hardware platforms, we used the metric of **reads per second**. To measure the quality of read alignments, from [11], we adopted the metric of sensitivity shown in Equation 4 and the metric of precision shown in Equation 5:

$$TP/(TP+FN) \quad (4)$$

$$TP/(TP+FP) \quad (5)$$

where TP, FP and FN are the number of true positives, false positives and false negatives, respectively. A true positive (for long reads) is a read mapped correctly that should be mapped (within 50-bp of the region [11]).

B. Workloads

To evaluate our FindeR, we adopted several state-of-the-art FM-Index-based genome analysis applications: BWA-MEM [17] for short and long read alignment, SGA [16] for short and long read assembly, Soap3 [27] for GPU-based short read alignment, ExactWordMatch [30] for genome annotation and a reference-based genome compression algorithm [31]. For long read alignment and assembly, we also applied the FM-Index-based error correcting technique, FMLRC [14], to reduce the number of errors.

C. Datasets

For genome alignment, annotation and compression, we used the chromosomes 1∼22, X and Y, from the latest human genome GRCh38 as the reference genome. To study FindeR on short reads, we used the Illumina platinum NA12878 human dataset [64] (ERR194147_1) consisting of 0.78G reads of 101-bp length with $50\times$ as the short read alignment input. To estimate the performance of FindeR on long reads, we created long reads (with length of 1K-bp) by PBSIM [65]. The error profiles of long reads [11] can be summarized in the format of (name, mismatch, insertion, deletion, total), e.g.,. (PacBio, 1.50%, 9.02%, 4.49%, 15.01%) and (ONT_2D, 16.50%, 5.10%, 8.40%, 30.0%). For *de novo* assembly, we used PBSIM and DWGSim [66] to generate long and short reads of *C. elegans* with $30\times$ coverage.

D. Schemes

To evaluate the speedup of all steps in genome analysis, we selected a 4GHz 8-core Intel Xeon E5-2667 (v2) processor. CPU power is measured by the RAPL Interface. To study the performance of FM-Index searches on various hardware platforms, we further chose a 1.3GHz 3840-cudaCore Nvidia Tesla P40 GPU with 24GB GDDR5, an FPGA implementation [33], and a $40nm$ ASIC chip [34]. GPU power is collected by the Nvidia visual profiler. To estimate the performance of short read alignment, we selected the ASIC designs including RaceLogic [22] and GenAx [12], and ReRAM content address memory (CAM)-based PIM architecture including RADAR [25], Bio-CAM [24] and R-CAM [21]. To evaluate the performance of long read alignment, we compared FindeR against a recent ASIC chip Darwin [11]. Since different accelerators are fabricated by various process technologies, we scaled all area and power metrics with $32nm$ technology.

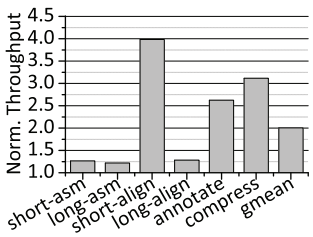


Fig. 14: Throughput.

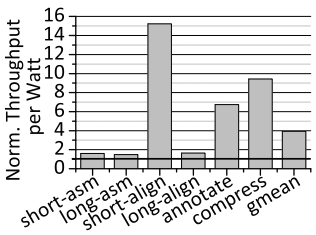


Fig. 15: Throughput per Watt.

V. Results and Analysis

A. FM-Index in Genome Analysis

We executed short and long read alignment, short and long read assembly, genome annotation and compression on our CPU baseline which can support all steps in genome analysis. We then moved the execution of FM-Index searches to FindeR. We report the throughput improvement of FindeR over the CPU in Figure 14 and the improvement of the throughput per Watt of FindeR over the CPU in Figure 15. During genome annotation and compression, FM-Index search is the most fundamental operation, accounting for $60\% \sim 70\%$ of CPU time. For short read alignment, only 0.3% of reads are processed by the SW algorithm. The bi-directional FM-Index search is used in both SMEM seeding and $\leq$ 2-mismatch seed extensions for the majority of short reads, so it costs 76% of CPU time. Therefore, compared to the CPU, FindeR improves the throughput of short read alignment, genome annotation and compression by $2.9\times$, $1.6\times$ and $2.1\times$ respectively. It increases the throughput per Watt of these steps by $14.2\times$, $5.7\times$ and $8.4\times$ respectively over our CPU baseline. For short read assembly, the FM-Index is used only during SMEM seeding. Long read assembly and alignment apply the FM-Index in SMEM seeding constructions and error corrections [14]. But the largest portion ($> 50\%$) of execution time during long read assembly and alignment is spent by the seed chaining filter [17] that does not invoke FM-Index searches. Compared to the CPU, FindeR improves the throughput of short and long read assembly, and long read alignment by 26.5%, 21.9% and 28.1% respectively. It boosts the throughput per Watt of these steps by 59.5%, 48.2% and 63.6% respectively over the CPU. On average, compared to the CPU, FindeR improves the throughput and the throughput per Watt of all steps in the genome sequencing by 101% and 294%, respectively.

	CPU	GPU [27]	ASIC [34]	FPGA [33]	**FindeR**
Die size (mm^2)	14.3K	1.6K	352	14.8K	1.1K
Main memory(GB)	128	6	1.3	48	0
Power(W)	130	258	3.1	247	9.09
Throughput	68K	150K	379K	1.5M	**10.7M**
Throughput/Watt	523	581	121K	6.2K	**1.18M**

TABLE I: FM-Index search on various hardware platforms.

B. FM-Index on Various Hardware Platforms

We used the SMEM seeding construction of short read alignment to evaluate FindeR and compare it against various hardware platforms, since the CPU-, GPU-, FPGA-, and ASIC-based FM-Index accelerators support the SMEM seeding construction where the FM-Index searches cost >97%

of the application execution time [20]. The FM-Index search throughput, area and power comparison of various hardware platforms is shown in Table I. Except our FindeR, all the other FM-Index search accelerators reply on the off-chip main memories to store FM-Index data and reads, and thus waste huge amounts of energy when fetching data from off-chip main memories. Since the CPUs, GPUs and FPGAs adopt software-based multi-step FM-Index optimizations [27], [33] to increase the FM-Index search throughput and fully utilize the hardware resources, they require a large capacity of DRAMs. The BWT of the FM-Index effectively compresses the genome reference data, so it is possible for the ASIC [34] to conduct FM-Index backward searches using only 1.3GB DRAM. Compared to the ASIC design, FindeR boosts the throughput and throughput per Watt of FM-Index searches by $28.2\times$ and $9.75\times$.

C. FM-Index for Short Read Alignment

1) Performance

To highlight the performance of FindR on short read alignment, we compared it against prior hardware accelerators [12], [21], [22], [24], [25] focusing on aligning short reads. These accelerators can boost the performance of either seeding [24], [25] or seed extension [12], [21], [22] for short reads. In contrast, only our FindeR is able to process both short read seeding by FM-Index searches and seed extension by FM-Index-based k-mismatch searches.

The 3D-ReRAM-based RADAR [25] can only process EPMs with no mismatches, while the 2D-ReRAM-based PIM BioCAM [24] can create seeds with $\leq$1 mismatch. Both of them rely on huge capacity ReRAM CAMs consuming significant power and area overhead. With the same capacity, compared to memory arrays, CAM increases the power and area by $> 10\times$ [58]. FindeR improves the short read seeding throughput and throughput per Watt by $20.7\times$ and $353.8\times$ over BioCAM. The more mismatches FindeR has to process, the more iterations it has to run. Therefore, FindeR attains 1179.4K, 707.7K and 424.6K 100-bp reads/sec/Watt with 0, 1 and 2 mismatches. When considering only EPMs, FindeR improves the short read seeding throughput per Watt by 117K$\times$ over 3D ReRAM-based RADAR. The significant improvement comes from two factors: First, the FM-Index algorithm of FindeR is much more efficient than the naïve exhaustive search adopted by RADAR; Second, the ReRAM CAMs have to consume more power than our RHUs, due to their huge peripheral circuits.

Both dynamic-programming-based accelerators (i.e., RaceLogic [22] and RCAM [21]) and the automata-based GenAx [12] require huge capacity for off-chip or internal memories to store complex intermediate data structures such as seed pointer tables, a position table, pointers, reads and references. They obtain optimal alignment results and can tolerate any number of mismatches by executing the high computational complexity SW algorithm. But FindeR improves the short read seed extension throughput and throughput per Watt by 83.8% and $4.9\times$ over RaceLogic, respectively.

2) Seed Quality

Among all prior accelerators, GenAx achieves the highest quality [12] of short read seeding, since it extends a k-bp hashed seed by a stride of $k/2^n$-bp until a SMEM is built, where $n = [1, \lfloor log_2(k) \rfloor]$. To evaluate the seed quality, we implemented the GenAx hash-table-based SMEM seeding to generate seeds, and used BWA-MEM to produce alignment mappings on our CPU baseline. Because GenAx uses $k = 12$ [12], the SMEM length of GenAx is often ≤ 24. FindeR creates SMEMs with an average length of 48. The alignment mappings built with the SMEM seeds generated by GenAx achieve 99.45% sensitivity and 99.71% precision. FindeR improves the sensitivity by 0.54% and the precision by 0.28%. Compared to GenAx, FindeR also reduces the hit number per read by 81% in short read alignment. A smaller hits/read means FindeR less frequently invokes computationally expensive FM-Index-based k-mismatch searches and reduces the seed extension time.

D. FM-Index for Long Read Alignment

We present the seeding results of long read alignment by comparing FindeR to a recent long read genomics co-processor Darwin [11] in Table III. Darwin proposes a novel hash-table-based seeding filter to reduce the seed extension overhead and a fast dynamic programming algorithm for APM in seed extensions. We used FindeR for long read SMEM seeding. Due to the large power-hungry SRAM buffers, Darwin consumes $15W$ power. So FindeR improves the throughput per Watt (the value having prefix "S:") by $5.3\times$ when only running for long read SMEM seeding. We implemented and executed the Darwin alignment flows on our CPU baseline to evaluate the seed quality. Compared to BWA-MEM and SGA, Darwin improves the sensitivity and precision of alignment and assembly mappings by $1\% \sim 5\%$ [11]. However, unlike short read alignment relying only on the seed-and-extend paradigm, long read alignment requires error corrections for high error rate long reads [14]. When the FM-Index-based error correction technique FMLRC [14] is applied, FindeR achieves slightly better sensitivity and precision ($< 0.1\%$) than Darwin for long read alignment. But when FindeR performs error correction and SMEM seeding concurrently for long reads, its throughput per Watt (the value having prefix "E:") improvement over Darwin degrades to 88%, due to the conflicts between error correcting and seeding requests inside each FindeR bank. Thanks to error corrections and SMEMs, FindeR decreases hits/read by $3\% \sim 10\%$ over Darwin.

VI. Prior art

Genome sequencing is the key to improving healthcare diagnoses, ensuring global food security and enforcing the wildlife conservation. But it is challenging to quickly and power-efficiently process such huge amounts of genomic data generated by genome sequencing machines. Application-specific acceleration for genome sequencing has become essential.

Both approximate pattern matching (APM) and exact pattern matching (EPM) are essential to genome sequencing, due to the seed-and-extension computing paradigm. Because of the high time complexity of APM algorithms for seed extensions,

	Hash Table		Dynamic Programming		Automata	FM-Index
	RADAR [25]	BioCAM [24]	Race [22]	RCAM [21]	GenAx [12]	**FindeR**
Die size (mm^2)	120	9.8K	450	383	4.6K	1.1K
Off-chip Memory(GB)	0	0	8	0	120	0
Power(W)	12.5	153	24.3	6.6K	20	9.09
Function	Seeding		Seed Extension			Both
Throughput	125	186.8K	2.1M	177K	973	**3.86M**
Throughput/Watt	10	1.2K	86K	26	48.65	**424.6K**

TABLE II: The comparison between various accelerators for short read alignment.

	Performance		Quality		
	Throughput	Throughput/Watt	hits/read	Sensitivity	Precision
Dar-Pac	3.9K	0.26K	0.33	99.71%	99.91%
Dar-ONT			0.45	98.2%	99.1%
Fin-Pac	S: 2.9K	S: 1.64K	0.29	99.8%	99.95%
Fin-ONT	E: 0.87K	E: 0.49K	0.44	98.31%	99.23%

TABLE III: The throughput and quality of long read seeding.

recent work presents ASIC- [11], [12], [22], FPGA- [23] and PIM-based [21], [24], [25] accelerators to enhance the APM performance during seed extensions. Particularly, several works [21], [24], [25] take advantage of power-hungry and low-density ReRAM-based content address memories (CAMs) to implement the SW algorithm to align short reads. However, compared to commodity ReRAM arrays, ReRAM CAMs with the same capacity enlarge the chip area and power consumption almost by $10\times$ [58].

In this paper, we show that compared to APMs, EPMs are more elementary and more time-consuming in most critical steps of genome analysis. During the short read seed extensions, FM-Index-based k-mismatch searches can even completely replace APMs, because the majority of sequencing errors are substitutions [37] (mismatches). For EPM operations, prior work only creates FPGA- [19], [20], [32], [33] and ASIC-based [34] accelerators to accelerate FM-Index searches notorious for massive random memory accesses. To the best of our knowledge, FindeR is the first PIM to accelerate FM-Index searches by ReRAM arrays.

VII. Conclusion

Enhancing the computing efficiency of genome analysis is urgent and important for personalized medical care, since it is projected that each individual's genome may be sequenced and analyzed over the next decade. Unlike previous genomics accelerators, we focus on the hardware acceleration of EPM during genome analysis. We propose a reliable and power-efficient ReRAM-based Hamming distance unit to accelerate the FM-Index-based EPM widely adopted in critical steps of genome analysis including genome assembly, alignment, annotation and compression. We further architect a full-fledged pipelined PIM, FindeR, with a system library to improve FM-Index search throughput. Compared to prior accelerators, FindeR improves the FM-Index search throughput by $83\% \sim 30K\times$ and throughput per Watt by $3.5\times \sim 42.5K\times$.

References

[1] M. Schirmer, R. D'Amore, U. Z. Ijaz, N. Hall, and C. Quince, "Illumina error profiles: resolving fine-scale variation in metagenomic sequencing data," *BMC bioinformatics*, vol. 17, no. 1, p. 125, 2016.

[2] J. J. Mosher, B. Bowman, E. L. Bernberg, O. Shevchenko, J. Kan, J. Korlach, and L. A. Kaplan, "Improved performance of the pacbio smrt technology for 16s rdna sequencing," *Journal of microbiological methods*, vol. 104, pp. 59–60, 2014.

[3] M. Eisenstein, "Oxford nanopore announcement sets sequencing sector abuzz," *Nature Biotechnology*, vol. 30, no. 4, pp. 295–297, 2012.

[4] J. D. Merker, A. M. Wenger, T. Sneddon, M. Grove, Z. Zappala, L. Fresard, D. Waggott, S. Utiramerur, Y. Hou, K. S. Smith *et al.*, "Long-read genome sequencing identifies causal structural variation in a mendelian disease," *Genetics in Medicine*, vol. 20, no. 1, p. 159, 2018.

[5] X. Ma, M. Mau, and T. F. Sharbel, "Genome editing for global food security," *Trends in biotechnology*, 2017.

[6] I. A. Arif, H. A. Khan, A. H. Bahkali, A. A. Al Homaidan, A. H. Al Farhan, M. Al Sadoon, and M. Shobrak, "Dna marker technology for wildlife conservation," *Saudi journal of biological sciences*, vol. 18, no. 3, pp. 219–225, 2011.

[7] E. Hanna, C. Rémuzat, P. Auquier, and M. Toumi, "Gene therapies development: slow progress and promising prospect," *Journal of market access & health policy*, vol. 5, no. 1, p. 1265293, 2017.

[8] S. Canzar and S. L. Salzberg, "Short read mapping: An algorithmic tour," *Proceedings of the IEEE*, vol. 105, no. 3, pp. 436–458, 2017.

[9] K. Davies, "Ring in the new: Rady children's hospital introduces novaseq for newborn diagnoses," *Clinical OMICs*, vol. 4, no. 6, pp. 16–17, 2017.

[10] Z. D. Stephens, S. Y. Lee, F. Faghri, R. H. Campbell, C. Zhai, M. J. Efron, R. Iyer, M. C. Schatz, S. Sinha, and G. E. Robinson, "Big data: Astronomical or genomical?" *PLoS Biology*, vol. 13, no. 7, 2015.

[11] Y. Turakhia, G. Bejerano, and W. J. Dally, "Darwin: A genomics co-processor provides up to 15,000x acceleration on long read assembly," in *ASPLOS*, 2018.

[12] D. Fuijiki, A. Subramaniyan, T. Zhang, Y. Zheng, R. Das, D. Blaauw, and S. Narayanasamy, "Genax: A genome sequencing accelerator," in *IEEE/ACM International Symposium on Computer Architecture*, 2018.

[13] J. Pevsner, *Bioinformatics and functional genomics*. John Wiley & Sons, 2015.

[14] J. R. Wang, J. Holt, L. McMillan, and C. D. Jones, "Fmlrc: Hybrid long read error correction using an fm-index," *BMC bioinformatics*, vol. 19, no. 1, p. 50, 2018.

[15] Y.-T. Huang and Y.-W. Huang, "An efficient error correction algorithm using fm-index," *BMC bioinformatics*, vol. 18, no. 1, p. 524, 2017.

[16] J. T. Simpson and R. Durbin, "Efficient de novo assembly of large genomes using compressed data structures," *Genome research*, vol. 22, no. 3, 2012.

[17] H. Li, "Aligning sequence reads, clone sequences and assembly contigs with bwa-mem," *arXiv preprint arXiv:1303.3997*, 2013.

[18] ——, "Exploring single-sample snp and indel calling with whole-genome de novo assembly," *Bioinformatics*, vol. 28, no. 14, pp. 1838–1844, 2012.

[19] E. J. Houtgast, V. M. Sima, K. Bertels, and Z. Al-Ars, "An fpga-based systolic array to accelerate the bwa-mem genomic mapping algorithm," in *IEEE International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation*, July 2015, pp. 221–227.

[20] M. C. F. Chang, Y. T. Chen, J. Cong, P. T. Huang, C. L. Kuo, and C. H. Yu, "The smem seeding acceleration for dna sequence alignment," in *IEEE International Symposium on Field-Programmable Custom Computing Machines FCCM*, 2016, pp. 32–39.

[21] R. Kaplan, L. Yavits, R. Ginosar, and U. Weiser, "A resistive cam processing-in-storage architecture for dna sequence alignment," *IEEE Micro*, 2017.

[22] A. Madhavan, T. Sherwood, and D. Strukov, "Race logic: A hardware acceleration for dynamic programming algorithms," in *IEEE/ACM International Symposium on Computer Architecture*, 2014.

[23] E. Rucci, C. Garcia, G. Botella, A. De Giusti, M. Naiouf, and M. Prieto-Matias, "Accelerating smith-waterman alignment of long dna sequences with opencl on fpga," in *International Conference on Bioinformatics and Biomedical Engineering*. Springer, 2017, pp. 500–511.

[24] S. K. Khatamifard, Z. I. Chowdhury, N. Pande, M. Razaviyayn, C. H. Kim, and U. R. Karpuzcu, "A non-volatile near-memory read mapping accelerator," *CoRR*, vol. abs/1709.02381, 2017.

[25] W. Huangfu, S. Li, X. Hu, and Y. Xie, "Radar: A 3d-reram based dna alignment accelerator architecture," in *IEEE/ACM Design Automation Conference*, 2018.

[26] P. Ferragina and G. Manzini, "An experimental study of an opportunistic index," in *ACM-SIAM Symposium on Discrete Algorithms*, 2001, pp. 269–278.

[27] R. Luo, T. Wong, J. Zhu, C.-M. Liu, X. Zhu, E. Wu, L.-K. Lee, H. Lin, W. Zhu, D. W. Cheung *et al.*, "Soap3-dp: fast, accurate and sensitive gpu-based short read aligner," *PloS one*, vol. 8, no. 5, p. e65632, 2013.

[28] M. Burrows and D. J. Wheeler, "A block-sorting lossless data compression algorithm," 1994.

[29] N. Ahmed, K. Bertels, and Z. Al-Ars, "A comparison of seed-and-extend techniques in modern dna read alignment algorithms," in *IEEE International Conference on Bioinformatics and Biomedicine*, 2016, pp. 1421–1428.

[30] J. Healy, E. E. Thomas, J. T. Schwartz, and M. Wigler, "Annotating large genomes with exact word matches," *Genome research*, vol. 13, no. 10, pp. 2306–2315, 2003.

[31] P. Prochazka and J. Holub, "Compressing similar biological sequences using fm-index," in *Data Compression Conference*, 2014, pp. 312–321.

[32] J. Arram, M. Pflanzer, T. Kaplan, and W. Luk, "Fpga acceleration of reference-based compression for genomic data," in *IEEE International Conference on Field Programmable Technology*, 2015, pp. 9–16.

[33] J. Arram, T. Kaplan, W. Luk, and P. Jiang, "Leveraging fpgas for accelerating short read alignment," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2017.

[34] Y. C. Wu, C. H. Chang, J. H. Hung, and C. H. Yang, "A 135-mw fully integrated data processor for next-generation sequencing," *IEEE Transactions on Biomedical Circuits and Systems*, 2017.

[35] A. Chacon, S. Marco-Sola, A. Espinosa, P. Ribeca, and J. C. Moure, "Boosting the fm-index on the gpu: Effective techniques to mitigate random memory access," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 12, no. 5, pp. 1048–1059, Sept 2015.

[36] A. Chacón, J. C. Moure, A. Espinosa, and P. Hernández, "n-step fm-index for faster pattern matching," *Procedia Computer Science*, vol. 18, pp. 70–79, 2013.

[37] M. A. Quail, M. Smith, P. Coupland, T. D. Otto, S. R. Harris, T. R. Connor, A. Bertoni, H. P. Swerdlow, and Y. Gu, "A tale of three next generation sequencing platforms: comparison of ion torrent, pacific biosciences and illumina miseq sequencers," *BMC genomics*, vol. 13, no. 1, p. 341, 2012.

[38] Y. Wang, X. Li, D. Zang, G. Tan, and N. Sun, "Accelerating fm-index search for genomic data processing," in *International Conference on Parallel Processing*, 2018.

[39] T. W. Lam, R. Li, A. Tam, S. Wong, E. Wu, and S. M. Yiu, "High throughput short read alignment via bi-directional bwt," in *IEEE International Conference on Bioinformatics and Biomedicine*, 2009.

[40] N. Ahmed, V. M. Sima, E. Houtgast, K. Bertels, and Z. Al-Ars, "Heterogeneous hardware/software acceleration of the bwa-mem dna alignment algorithm," in *IEEE/ACM International Conference on Computer-Aided Design*, Nov 2015, pp. 240–246.

[41] H. S. P. Wong, H. Y. Lee, S. Yu, Y. S. Chen, Y. Wu, P. S. Chen, B. Lee, F. T. Chen, and M. J. Tsai, "Metal-oxide rram," *Proceedings of the IEEE*, 2012.

[42] N. Ge, J. H. Yoon, M. Hu, E. J. Merced-Grafals, N. Davila, J. P. Strachan, Z. Li, H. Holder, Q. Xia, R. S. Williams, X. Zhou, and J. J. Yang, "An efficient analog hamming distance comparator realized with a unipolar memristor array: a showcase of physical computing," *Scientific Reports*, 2017.

[43] S. Ito, Y. Hayakawa, Z. Wei, S. Muraoka, K. Kawashima, H. Kotani, K. Kouno, M. Nakamura, G. A. Du, J. F. Chen *et al.*, "Reram technologies for embedded memory and further applications," in *IEEE International Memory Workshop*, 2018.

[44] C.-F. Lee, H.-J. Lin, C.-W. Lien, Y.-D. Chih, and J. Chang, "A 1.4 mb 40-nm embedded reram macro with 0.07 um 2 bit cell, 2.7 ma/100mhz low-power read and hybrid write verify for high endurance application," in *IEEE Asian Solid-State Circuits Conference*, 2017, pp. 9–12.

[45] M. Mao, Y. Cao, S. Yu, and C. Chakrabarti, "Optimizing latency, energy, and reliability of 1t1r reram through cross-layer techniques," *IEEE JETC*, 2016.

[46] C. Xu, D. Niu, N. Muralimanohar, R. Balasubramonian, T. Zhang, S. Yu, and Y. Xie, "Overcoming the challenges of crossbar resistive memory architectures," in *HPCA*, 2015.

[47] W. Wen, L. Zhao, Y. Zhang, and J. Yang, "Speeding up crossbar resistive memory by exploiting in-memory data patterns," in *ICCAD*, 2017.

[48] S. Schechter, G. H. Loh, K. Strauss, and D. Burger, "Use ecp, not ecc, for hard failures in resistive memories," in *ISCA*, 2010.

[49] D. Lee, J. Lee, M. Jo, J. Park, M. Siddik, and H. Hwang, "Noise-analysis-based model of filamentary switching reram with zro_x/hfo_x stacks," *IEEE Electron Device Letters*, vol. 32, no. 7, pp. 964–966, 2011.

[50] J. K. Park, S. Y. Kim, J. M. Baek, D. J. Seo, J. H. Chun, and K. W. Kwon, "Analysis of resistance variations and variance-aware read circuit for cross-point reram," in *IEEE International Memory Workshop*, 2013.

[51] T. Semiconductor, "Soft errors in electronic memory-a white paper," *White Paper*, 2004.

[52] L. Kull, T. Toifl, M. Schmatz, P. A. Francese, C. Menolfi, M. Braendli, M. Kossel, T. Morf, T. M. Andersen, and Y. Leblebici, "A 3.1mw 8b 1.2gs/s single-channel asynchronous sar adc with alternate comparators for enhanced speed in 32nm digital soi cmos," in *ISSCC*, 2013.

[53] M. Poremba, T. Zhang, and Y. Xie, "Fine-granularity tile-level parallelism in non-volatile memory architecture with two-dimensional bank subdivision," in *DAC*, 2016, pp. 1–6.

[54] T. J. Ham, B. K. Chelepalli, N. Xue, and B. C. Lee, "Disintegrated control for energy-efficient and heterogeneous memory systems," in *HPCA*, 2013.

[55] K. Fang, L. Chen, Z. Zhang, and Z. Zhu, "Memory architecture for integrating emerging memory technologies," in *PACT*, 2011.

[56] Y. Liu, T. Hankeln, and B. Schmidt, "Parallel and space-efficient construction of burrows-wheeler transform and suffix array for big genome data," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 13, no. 3, pp. 592–598, May 2016.

[57] B. Liu, D. Zhu, and Y. Wang, "debwt: parallel construction of burrows–wheeler transform for large collection of genomes with de bruijn-branch encoding," *Bioinformatics*, vol. 32, no. 12, pp. i174–i182, 2016.

[58] S. Li, L. Liu, P. Gu, C. Xu, and Y. Xie, "Nvsim-cam: A circuit-level simulator for emerging nonvolatile memory based content-addressable memory," in *IEEE/ACM International Conference on Computer-Aided Design*. ACM, 2016, p. 2.

[59] W. H. Choi, M. Lueker-Boden, M. Grobis, N. Robertson, and Z. Bandic, "A comprehensive study on ddr4 mram and reram power estimation using a parameterized nvm power calculator," in *IMW*, 2018.

[60] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," in *ACM/IEEE International Symposium on Computer Architecture*, 2016, pp. 27–39.

[61] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," in *ACM/IEEE International Symposium on Computer Architecture*, 2016, pp. 14–26.

[62] L. Song, X. Qian, H. Li, and Y. Chen, "Pipelayer: A pipelined reram-based accelerator for deep learning," in *IEEE International Symposium on High Performance Computer Architecture*, 2017, pp. 541–552.

[63] M. Poremba, T. Zhang, and Y. Xie, "Nvmain 2.0: A user-friendly memory simulator to model (non-)volatile memory systems," *CAL*, vol. 14, no. 2, pp. 140–143, Jul. 2015.

[64] M. A. Eberle, E. Fritzilas, P. Krusche, M. Källberg, B. L. Moore, M. A. Bekritsky, Z. Iqbal, H.-Y. Chuang, S. J. Humphray, A. L. Halpern *et al.*, "A reference data set of 5.4 million phased human variants validated by genetic inheritance from sequencing a three-generation 17-member pedigree," *Genome research*, 2016.

[65] Y. Ono, K. Asai, and M. Hamada, "Pbsim: Pacbio reads simulatortoward accurate genome assembly," *Bioinformatics*, vol. 29, no. 1, pp. 119–121, 2012.

[66] H. Li, B. Handsaker, A. Wysoker, T. Fennell, J. Ruan, N. Homer, G. Marth, G. Abecasis, and R. Durbin, "The sequence alignment/map format and samtools," *Bioinformatics*, vol. 25, no. 16, pp. 2078–2079, 2009.