GeNVoM: Read Mapping Near Non-Volatile Memory

S. Karen Khatamifard, Zamshed Chowdhury, Nakul Pande, Meisam Razaviyayn, Chris Kim, and Ulya R. Karpuzcu

Abstract—DNA sequencing is the physical/biochemical process of identifying the location of the four bases (Adenine, Guanine, Cytosine, Thymine) in a DNA strand. As semiconductor technology revolutionized computing, modern DNA sequencing technology (termed Next Generation Sequencing, NGS) revolutionized genomic research. As a result, modern NGS platforms can sequence hundreds of millions of short DNA fragments in parallel. The sequenced DNA fragments, representing the output of NGS platforms, are termed *read*s. Besides genomic variations, NGS imperfections induce noise in *reads*. Mapping each *read* to (the most similar portion of) a reference genome of the same species, i.e., *read mapping*, is a common critical first step in a diverse set of emerging bioinformatics applications. Mapping represents a search-heavy memory-intensive similarity matching problem, therefore, can greatly benefit from near-memory processing. Intuition suggests using fast associative search enabled by Ternary Content Addressable Memory (TCAM) by construction. However, the excessive energy consumption and lack of support for similarity matching (under NGS and genomic variation induced noise) renders direct application of TCAM infeasible, irrespective of volatility, where only *non-volatile* TCAM can accommodate the large memory footprint in an area-efficient way. This paper introduces GeNVoM, a scalable, energy-efficient and high-throughput solution. Instead of optimizing an algorithm developed for general-purpose computers or GPUs, GeNVoM rethinks the algorithm and non-volatile TCAM-based accelerator design together from the ground up. Thereby GeNVoM can improve the throughput by up to 3.67×; the energy consumption, by up to 1.36×, when compared to an ASIC baseline, which represents one of the highest-throughput implementations known.

Index Terms—Hardware accelerator, in-memory computing, content-addressable memory, bioinformatics, read mapping accelerator, DNA sequencing in memory

1 Introduction

DNA sequencing is the physical or biochemical process of extracting the order of the four bases (Adenine, Guanine, Cytosine, Thymine) in a DNA strand. As semiconductor technology revolutionized computing, DNA sequencing technology, termed *High-throughput Sequencing* or *Next Generation Sequencing*(NGS), revolutionized genomic research. As a result, modern NGS platforms can sequence hundreds of millions of short DNA fragments in parallel. The sequenced fragments (which represent the NGS output) are referred to as short *read*s and typically contain 100-200 bases [1]. The focus of this paper is *read mapping*, a common critical first step spanning a rich and diverse set of emerging bioinformatics applications: mapping each NGS *read* to (the most similar portion of) a reference genome of the same species (which itself is a full-fledged assembly of already processed *read*s).

As a representative example, modern NGS machines from Illumina [1], a prominent NGS platform producer, can sequence more than 600Giga-bases (Gba) per one run, 200× the length of a human genome of approximately 3Gba, which translates into hundreds of millions of output *reads*. Fig. 1 depicts the scaling trend in terms of the total number of human genomes sequenced. The values until 2015 reflect historical publication records, with milestones explicitly marked. The values beyond 2015 reflect three different projections: the first, following the historical growth until 2015; the second, a more conservative prediction from Illumina; the third, Moore's Law. Historically, the total quantity has been doubling approx. every 7 months. Even the more conservative projections from Fig. 1 (i.e., 2× increase every 12 or 18 months) result in a very rapid growth, which challenges the throughput performance of *read mapping*.

The wildly increasing scale of the problem per Fig. 1 renders well-studied pair-wise similarity detection algorithms inefficient [3]. Worse, *read*s are subject to noise due to imperfections in NGS platforms and genomic variations, which adds to the complexity. Both algorithmic solutions and hardware acceleration via GPUs [4], FPGAs [5], ASICs [6] therefore have to trade mapping accuracy for throughput performance. In other words, *read mapping* by definition is after *similarity matching*. As optimizations are usually confined to compute-intensive stages of mapping, considering scaling projections from Fig. 1, most of these solutions are fundamentally limited by data transfer overheads. In this paper, we instead take a data-centric position to guide the design. Specifically, instead of optimizing an algorithm developed for general-purpose computers or GPUs, we rethink the algorithm from the ground up along with the accelerator design.

- *S. Karen Khatamifard, Zamshed Chowdhury, Nakul Pande, Chris Kim, and Ulya R. Karpuzcu are with the Department of Electrical Engineering, University of Minnesota, Minneapolis, MN 55455 USA. E-mail: {khatami, chowh005, nakul, chriskim, ukarpuzc}@umn.edu.*
- *Meisam Razaviyayn is with the Department of Industrial & Systems Engineering, University of Southern California, Los Angeles, CA 90007 USA. E-mail: razaviya@usc.edu.*

Manuscript received 4 Dec. 2019; revised 18 Mar. 2021; accepted 4 Sept. 2021. Date of publication 6 Oct. 2021; date of current version 8 Dec. 2022. (Corresponding author: S. Karen Khatamifard.) Digital Object Identifier no. 10.1109/TCBB.2021.3118018

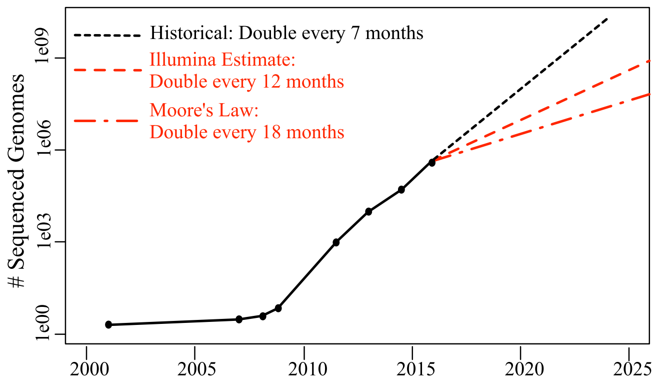


Fig. 1. Scaling trend for DNA sequencing [2].

Read mapping represents a search-heavy memory intensive operation and barely requires complex floating point arithmetic, therefore, can greatly benefit from near-memory processing. Intuition suggests using fast parallel associative search, enabled by Ternary Content Addressable Memory (TCAM) by construction, in matching short *read* patterns with portions of the large reference genome (stored in TCAM). As we will explain in Section 3, however, only *non-volatile* TCAM can accommodate the large memory footprint in an area- and energy-efficient manner [7]. Even then, brute-force non-volatile TCAM search over as large of a search space as *read mapping* demands induces excessive energy consumption, rendering (non-volatile) TCAM-based acceleration infeasible. At the same time, by construction, (non-volatile) TCAM cannot handle similarity matching under NGS or genomic variation induced noise.

This paper provides an effective solution, GeNVoM, to tap the potential of *non-volatile* TCAM for scalable, energy-efficient high-throughput *read mapping*. GeNVoM

- Explores similarity matching for resistive non-volatile TCAM (which can only handle exact matches by construction), to trade mapping accuracy for throughput and energy efficiency in a more scalable manner than existing solutions;
- Features a novel genomic data representation for similarity matching without compromising space complexity;
- Tailors common search space pruning approaches to its similarity matching mechanism in order to identify and discard unnecessary non-volatile TCAM accesses (and thereby, to prevent excessive energy consumption);
- Accounts for the most prevalent manifestations of noise induced by NGS imperfections and genomic variations during similarity matching, including (base) gaps and insertions/deletions in *reads*;
- Employs multi-phase hierarchical similarity matching to enhance mapping accuracy and scalability, where each phase performs progressively more sophisticated mapping, considering only the subset of *reads* the previous phase fails to map.

In the following, Section 2 provides a compare and contrast to related work; Section 3 discusses basics; Section 4 covers implementation details; Sections 5 and 6 detail the evaluation; and Section 7 summarizes our findings.

2 Related Work

(Short) Read Mapping. With no pre-processing of the *reference*, the computational complexity scales (at least) linearly with the *reference* length. Therefore, popular software such as SOAP [8], Eland (part of the Illumina suite), and MAQ [9] adapt hash-based pre-processing. SOAP2 [10], Bowtie (2) [11], [12], BWA [13], [14], on the other hand, use the (more memory efficient) Burrows-Wheeler Transformation (BWT).

One of the baselines, SOAP3-dp [15], is an open-source GPU implementation of BWA, which can also handle noise in *reads*, however, unlike GeNVoM, the computational complexity depends on the *tolerance*. GenAx [6], the other baseline for comparison, is an automata-based accelerator, where, similar to SOAP3-dp, computational complexity depends on the *tolerance*. GenAX also has two main units; seeding equivalent to FilterU (Section 3.3), and scoring equivalent to MatchU (Section 3.3), which can handle indels directly. As we show in Section 6, GenAX, relying on a complex, high-overhead alternative of GeNVoM's MatchU, is fundamentally slower than GeNVoM, which features a less complex MatchU optimized for the common-case. [16] shows a successful fabrication of GenAx in 55nm technology. AIM [17], another accelerator for computational genomics, only accelerates BWT kernel, which alone is not enough for an end-to-end read mapping.

High-throughput FPGA implementations [18] also exist, at the expense of orders of magnitude higher power consumption than GeNVoM. The exotic race logic based dynamic programming accelerator [19] can find the similarity between two strings corresponding to the *read* and a part of the *reference* in approx. 120ns while consuming 1nJ. This is much slower than GeNVoM, and energy grows with the third power of *read* length which can impair scalability. [20] also presents an accelerator based on dynamic programming. Recent work also includes a BWA-based hardware accelerator featuring NVM [21] (which is significantly slower than GeNVoM); and very effective hardware acceleration for other significant bioinformatics algorithms [22]. Another recent accelerator features a down-stream application, indel realignment, on FPGA at cloud-scale, leading to 81× speed-up [23].

Efficient filtering techniques for hash-based *read mapping* using 3D-stacked memories [24] and FPGAs[25], and alignment verification techniques [26] (to avoid expensive alignments for too different reads) also exist. While the high cost of quadratic-time dynamic programming algorithms for string (i.e., *query*) matching motivated these works, GeNVoM relies on much faster and more energy-efficient string matching enabled by resistive TCAM. Hence, GeNVoM employs a simpler, low-latency filtering (incorporated in FilterU), which is tailored to its more efficient string matching (incorporated in MatchU). Another recent accelerator features pre-alignment for longer reads employing a novel resistive approximate similarity search mechanism[27].

Resistive CAM Accelerators. Guo *et al.* [28], [29] explore TCAM for accelerating data-intensive applications. Yavits *et al.* [30] propose an associative processor, which employs resistive CAM based look-up tables to implement diverse functions. Kaplan *et al.* [31] demonstrate how to efficiently implement Smith-Waterman algorithm (which tries to align

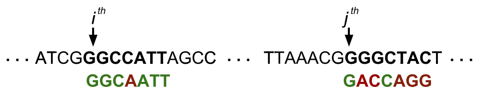


Fig. 2. Read mapping example.

strings of similar length) using resistive CAM based look-up tables. Not being able to handle similarity matching under noise, however, neither of these designs are directly applicable to *read mapping*.

Approximate resistive CAM is also proposed, either using exotic cells [32] (which complicates match detection, and thereby restricts the row length to at most 8 bits) or limiting the row length to 4 bits only [33]. While these show great potential, restricted row length hinders applicability to *read mapping* where longer (short) *read*s are emerging with NGS improvements and where TCAM search happens at row-granularity. GeNVoM feautures much less intrusive support for approximate matches due to tunable matching in a variation-aware manner, without restricting the row size. At the same time, no CAM array capable of only approximate matching would be sufficient to implement an efficient *read mapping* accelerator by itself, as demonstrated in Section 3.2.

Recent representative demonstrations of resistive TCAM include a 1Mbit PCM-based CAM from IBM using IBM 90nm technology [7] with a measured search latency of 1.9ns, and two novel spintronic designs in 45nm [34], where a search takes $\approx$0.6ns in 256-wide rows. Fundamentally, GeNVoM does not require any specific resistive technology, therefore can adapt any resistive CAM array, including these proposals or ReRAM based CAM [35].

Applications in Bioinformatics. DNA Methylation– a well-known epigenetics marker that is responsible for modifying gene expression, may cause cancer. [36] finds methylation-disease path associations with diseases such as ovarian cancer through network-guided association mapping. [37] proposes a novel supervised dimensionality reduction which is useful to perform data visualization and pattern mining (e.g., disease classification using gene expression). Transcription factor binding site is an area in DNA sequence, responsible for controlling transmission of genetic information from DNA to messenger RNA. [38] proposes using recurrent neural networks to identify transcription factor binding sites from DNA sequence data. It is important to understand the drug resistance of Mycobacterium tuberculosis bacteria (or, MTB) against available TB drugs– in order to handle the global epidemic effectively. [39] uses different classifiers on DNA sequence data, and compares the performance with existing baseline techniques. Identifying post translational modification sites is important in understanding key biological regulations. [40] identifies the modification sites with a classifier that uses general Pseudo Amino Acid Composition as features. [41] presents an ensemble of different types of neural network algorithms to identify the post translational modification sites which is applicable to different types of modification sites. [42] and [43] employ a flexible neural tree as a classification model to accurately identify the modification types of lysine malonylation– a key protein post translational modification. A polynomial tree method, on top of the flexible neural tree model, is used by [44] for identification of Lysine Acetylation site. On the other hand, [45] utilizes a multi-layer neural network and support vector machine to predict the potential modified sites.

3 GeNVoM: Macroscopic View

Scope. Currently, short (i.e., 100-200 base long) *read*s from modern Illumina NGS platforms [1] constitute more than 90% of all *read*s in the world. Accordingly, GeNVoM is designed for short *read* mapping.

Terminology. Without loss of generality, we will refer to each *read* as a *query*; and the reference genome, as the *reference*. Each *query* and the *reference* represent strings of characters from the alphabet {A, G, C, T} which stand for the bases {Adenine, Guanine, Cytosine, Thymine}. The inputs to GeNVoM are a dataset of *query*s and the *reference*, where the *reference* is many orders of magnitude longer than each *query*. For example, if the *reference* is the human genome, *reference* length is approximately 3×10^9 bases. On the other hand, technological capabilities of modern NGS platforms limit the maximum *query* length.

3.1 Problem Definition: Read Mapping

Basics. Read mapping entails finding the most *similar* portions of a given *reference* to each *query* from a dataset corresponding to the same species, as output by an NGS machine. Fig. 2 demonstrates an example, with different portions from the same *reference* on top; two sample *query*s to be mapped, at the bottom. The first (second) *query* results in one (five) base-mismatch(es) when aligned to the i^{th} (j^{th}) base of the *reference*. The *query* length is not representative, but simplifies demonstration.

GeNVoM's input *query*s are subject to noise due to imperfections in NGS platforms and potential genomic variations. Therefore, *read mapping* by definition is after *similarity* rather than an *exact match*. Hence, for each input *query*, GeNVoM locates the *most similar* sub-sequence of the *reference* to the *query*, and returns the range of its indices.

Mapping Reverse Complement of Reads. It is not uncommon for NGS platforms to sequence DNA strands in reverse direction. This happens when sequencing starts from the last base of a DNA strand. In this case, the platform outputs the reverse complement of a *read* by interchanging A with T, and C with G. For example, the reverse complement of the sequence ACCGCCTA is TAGGCGGT. NGS platforms typically sequence almost half of the DNA strands in reverse order, hence, GeNVoM is designed to handle these *read*s.

Sources of Noise in Similarity Matching. In general, the sequenced genome (where the *reads* are coming from) is expected to be slightly different from the *reference* genome, even though they represent the very same species. *Genomic variations* induce such differences, which can lead to base-mismatches between the *query*s and the *reference*, since the *query*s come from the sequenced genome as opposed to the *reference*. NGS platform imperfections, as well, can result in false base-mismatches between the *query*s and the *reference*, due to the so-called *read errors* during sequencing. We will next discuss the most prevalent manifestations of genomic variations and read errors.

Noise Manifestation. Most common genomic variations and read errors manifest themselves in three ways: Random *insertion* of a base, random *deletion* of a base, and random

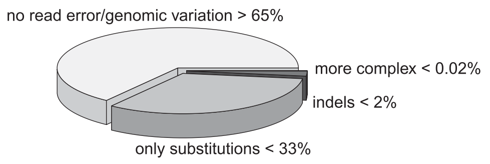


Fig. 3. Manifestation of genomic variations & read errors.

substitution of a base with another. Insertions and deletions are often referred to as *indels*. The expected rates of indels and substitutions depend on the type of genomes (hence species) and the NGS technology.

As a representative example, Fig. 3 demonstrates the typical share of *read*s having no read errors/genomic variations ($>$65%), at least one substitution (and no indels) ($<$33%), at least one indel ($<$2%), and more complex manifestations ($<$0.02%) [46], [47], [48], [49]. Mapping under rare complex manifestations (such as long-indels, gaps, base duplications or inversions) is a daunting task, and to date there is no widely-accepted algorithm to cover all [48], [50]. As Fig. 3 shows, substitutions are dominant. Although indel rate is on the lower side, detecting indels is critical for many downstream applications. Hence, GeNVoM is designed to cover both substitutions and indels, but optimized for the common case (no read error/genomic variation and only substitutions), which covers more than 98% of the *read*s per Fig. 3. As we will demonstrate in Section 4.4, straight-forward expansion of GeNVoM to more complex manifestations such as gaps is also possible.

3.2 Why Naive (Non-Volatile) TCAM-Based Acceleration Does not Work

Read mapping essentially is a search-heavy memory intensive pattern matching problem. This suggests TCAM-based acceleration, which by construction can support fast parallel in-memory search. TCAM is a special variant of associative memory (which permits data retrieval by indexing by content rather than by address) that can store and search the "don't care" state X in addition to a logic 0 or 1. Considering the scale of the problem, however, only *non-volatile* TCAM can accommodate the large memory footprint in an area- and energy-efficient manner [7].

We will next look into the energy consumption of *read mapping*, comparing a non-volatile TCAM-based implementation with a highly optimized GPU-based solution deploying one of the fastest known algorithms to date [15]. The non-volatile TCAM design from [7] features an array size of 256$\times$256 = 64Kbits. For this design point, searching for a pattern of length 256bits (which represents the maximum-possible length, i.e., the row length) in the entire array takes approximately 0.9ns and consumes 15.3nJ. If we simply encode each base from the alphabet {A, G, C, T} using 2 bits, and if a human genome of approximately 3Giga-bases (=6Gbits) represents the *reference*, the *reference* can fit into 91.6K TCAM arrays (of size 256 $\times$ 256bits = 64Kbits).

For each *query* of a typical length of 100 bases [1], i.e., 200 bits, the following naive procedure can then cover the entire search space: By construction, each 256$\times$256bit TCAM array can search for at most one 256bit pattern at a time, which resides in a query register. We can align the most significant bit of the (200bit-long) *query* with the most significant bit position of TCAM's 256bit query register, and pad the remaining (256-200) bits by Xs, for the very first search in the array. We can then repeat the search by shifting the contents of TCAM's query register (i.e., the padded *query*) to the right by one bit at a time, leaving the unused more significant bit positions with Xs, until the least significant bit of the *query* reaches the least significant bit position in the query register. The total number of these bit-wise shifts (and hence, searches) would be in the order of the row length $\approx$ 256. Putting it all together, mapping a given *query* to the *reference* in this case would take around 256 searches in each of the 91.6K arrays, with 15.3nJ consumed per search. The overall energy consumption therefore would become 91.6K $\times$ 256 $\times$ 15.3nJ $\approx$ 358.8mJ.

The GPU solution from Luo *et al.* [15] on the other hand, can process 133.3K *query*s per second. Hence, it takes 1/133.3K seconds to map a single *query*. Even under the unrealistic assumption (favoring TCAM) that the entire peak average power (TDP) goes to mapping a single *query* to the *reference*, the energy consumption would become at most 235W $\times$ (1/133.3K)s $\approx$ 1.8mJ.

The GPU and TCAM designs feature similar technology nodes, however, even by favoring TCAM, the TCAM-based naive implementation consumes approx. 2 orders of magnitude more energy than the GPU-based. This difference stems from the gap in the size of the search spaces. While the TCAM-based design considers the entire search space to cover all possible alignments, the GPU-based design first prunes the search space to eliminate infeasible alignments, which in turn leads to orders of magnitude less number of (search) operations. GeNVoM, while deploying non-volatile TCAM arrays, adopts a similar pruning strategy to enable more energy-efficient search.

Even if excessive energy consumption was not the case, (non-volatile) TCAM has another fundamental limitation which hinders applicability to *read mapping*. As we will detail in Section 4.3, even in the presence of "don't cares", TCAM cannot handle similarity matching considering various manifestations of noise due to NGS errors and genomic variations (Section 3.1).

To summarize, both, *the excessive energy consumption and lack of support for similarity matching render a direct adaption of non-volatile TCAM-based search infeasible*. The energy overhead of conventional volatile TCAM would be even higher [7], while the restriction on similarity matching directly applies irrespective of volatility.

GeNVoM unlocks the throughput potential of non-volatile TCAM in a scalable and energy-efficient manner through

- a *non-volatile* resistive TCAM design capable of *similarity matching* (Section 4.3);
- a novel *genomic data representation for similarity matching* without compromising storage complexity (Section 4.2);
- a common filtering mechanism [51] for *search space pruning adapted to non-volatile similarity matching* to prevent excessive energy consumption (Section 4.1);
- *hierarchical similarity matching* to maximize mapping accuracy without compromising scalability (Section 4.4).

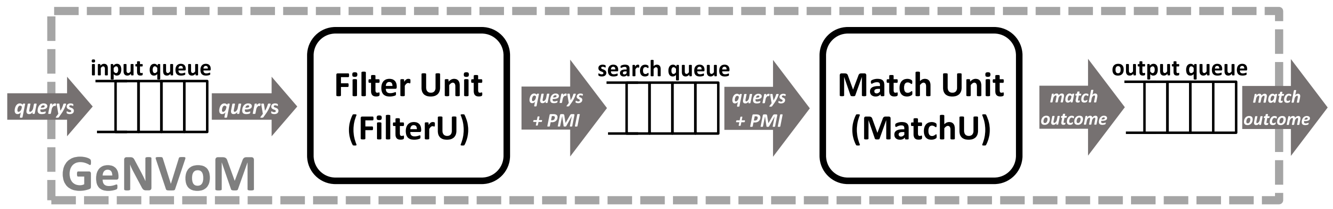


Fig. 4. Structural organization.

Designed for similarity search (in the presence of NGS or genomic variation triggered noise), GeNVoM's non-volatile TCAM arrays can directly handle substitutions, by construction (Section 4.3). To cover indels and more complex corruptions, on the other hand, GeNVoM adapts *anchoring* within its multi-phase mapping hierarchy (Section 4.4). The key insight is that complex corruptions that can lead to failed mappings are much less likely to occur in *all* portions of a *read* simultaneously. This makes anchoring very effective – a divide and conquer technique which entails chunking the *read* (at an anchored base position) to shorter substrings and attempting mapping on each chunk simultaneously. Thereby, problematic chunk(s) (and hence the entire *read*) simply follow the alignment dictated by the less-corrupted chunks, which by construction renders the most accurate mapping under noise. Numerous prevalent *read mapping* algorithms [52] therefore use anchoring-based techniques for complex corruptions.

3.3 Hardware Organization

Fig. 4 provides the structural organization. GeNVoM pipeline comprises two major units: Filter Unit (FilterU) and Match Unit (MatchU). Each *query* from the dataset to be mapped streams into the (first stage of the) GeNVoM pipeline (i.e., FilterU) over the *input queue*. Once the mapping completes, the outcome streams out of the (last stage of the) GeNVoM pipeline (i.e., MatchU) over the *output queue*. Non-volatile TCAM arrays (featuring GeNVoM's novel similarity matching mechanism) within MatchU keep the entire *reference*.

Input and output queues handle the communication to the outside world, by retrieving *query*s on the input end, and upon completion of the mapping, by providing the indices of the most *similar* sub-sequences of the *reference* to each *query*, on the output end.

FilterU filters (indices of) sub-sequences of the *reference* which are more likely to result in a match to the incoming *query*, by examining sub-sequences of the incoming *query* itself. We call these indices *potentially matching indices*, PMI. FilterU feeds the MatchU with a stream of $<$PMI, *query*$>$ tuples over the *search queue*. MatchU in turn conducts the search by only considering PMI of the *reference*. In this manner, GeNVoM prunes the search space.

The input queue feeds the GeNVoM pipeline with the *query*s to be mapped to the *reference*. The *query* dataset resides in memory. GeNVoM initiates the streaming of the *query*s into the memory-mapped input queue over a Direct Memory Access (DMA) request. The input queue in turn sends the *query*s to FilterU for search space pruning before the search takes place. Finally, for each *query*, once the mapping completes, the output queue collects from MatchU the indices of the sub-sequence of the *reference* featuring the most similar match to the *query*. The output queue is memory-mapped, too. GeNVoM writes back these indices to a dedicated memory location over DMA.

In the following, we will detail the steps for *query* processing in each unit, in case of a match. If no sub-sequence of the *reference* matches the input *query*, no mapping takes place, and GeNVoM updates a dedicated flag at the memory address to hold the result. GeNVoM can detect such failed mapping attempts during processing at FilterU or at MatchU.

Filter Unit. Fig. 5 provides the structural organization of FilterU, which prunes the search space as follows: We will refer to each sub-sequence of length *seed* as a *prefix*, where *seed* represents a design parameter and typically assumes a much lower value than the *query* length. As each *prefix* is a string of characters from the 4-character alphabet {A, G, C, T}, a *prefix* of length *seed* can take 4^{seed} different forms. Considering the size of the problem, the *reference* is likely to occupy multiple TCAM arrays. FilterU relies on a pre-processing step which entails identifying each *prefix* of length *seed* in the *reference*, and recording the TCAM array, column and row number of the corresponding occurrence. *Potential Match Index Table* PMIT keeps this information.

However, as the same *prefix* may occur multiple times along the *reference* string, PMIT may contain multiple entries for the very same *prefix*. Therefore, FilterU has another table called *PMIT Index Locator* (PMITIL) for bookkeeping. PMITIL serves as a dictionary of 4^{seed} entries, considering all possible 4^{seed} values of the (*seed*-long) *prefix*. Each PMITIL entry corresponds to a specific *prefix* value, and keeps the start address in PMIT where the TCAM indices for the corresponding occurrence of the *prefix* (along the *reference*) reside. As PMIT is organized to keep multiple occurrences (along the *reference*) of the same *prefix* consecutively, it suffices to keep per PMITIL entry just the start address (in the PMIT) for the first occurrence. The end address in this case simply is the start address stored in the next PMITIL entry.

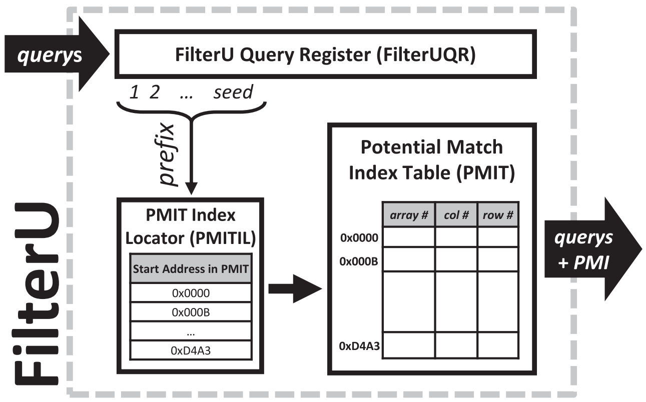


Fig. 5. Filter Unit (FilterU).

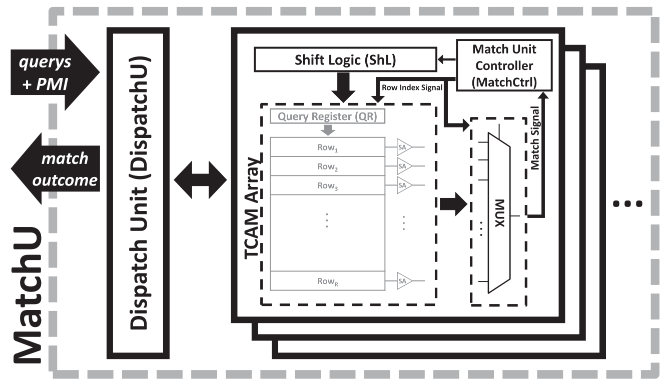


Fig. 6. Match Unit (MatchU).

PMIT and PMITIL generation constitutes a pre-processing step which GeNVoM needs to perform only once, offline, for each *reference*, before *read mapping* starts. As *read mapping* entails mapping a large number short *read*s to a given *reference* of the same species, this overhead does not apply to runtime, and is easy to amortize.

Upon receipt of a new *query* from the head of the input queue, FilterU uses the first *seed* bases of the *query* as the *prefix* to consult PMITIL, and subsequently, PMIT. FilterU keeps the *query* being processed in the FilterU *Query Register* (FilterUQR) as filtering is in progress. If there is a match in the PMI tables, FilterU first broadcasts the *query* being processed to all TCAM arrays. Then, it sends the corresponding TCAM array, column and row number (i.e., the Potential Match Indices, PMIs) to MatchU, over the *search queue*. We will refer to these TCAM coordinates as $array_{\#}$, $col_{\#}$, and $row_{\#}$, respectively.

Match Unit. Fig. 6 provides the structural organization of MatchU, which orchestrates search. MatchU features the *Dispatch Unit* (DispatchU) and non-volatile TCAM arrays capable of similarity search under NGS or genomic variation induced noise. DispatchU acts as a scheduler for TCAM search. For each input *query* to be mapped to the *reference*, DispatchU collects TCAM $array_{\#}$, $col_{\#}$ and $row_{\#}$, extracted from the PMIT in FilterU, to initiate the targeted search.

The input *query* stays in the *Query Register* (QR) of the TCAM array $array_{\#}$ during TCAM access. *Shift Logic* (ShL) in TCAM array $array_{\#}$ in turn first aligns the *prefix* of length *seed* of the *query* with the *seed*-long (matching) sub-sequence of the *reference* residing (in array $array_{\#}$) in row $row_{\#}$, starting from column $col_{\#}$. To this end, ShL shifts *query* bits in QR and inserts Xs accordingly. *Match Unit Controller* (MatchCtrl) orchestrates this operation. Once alignment completes, MatchCtrl activates the row $row_{\#}$ for search. Once the search completes, MatchCtrl provides DispatchU with the indices of the *reference* which demarcate the most similar sub-sequence to the entire *query*. DispatchU then forwards these indices to the output queue.

Fig. 8 depicts two different match scenarios: In Fig. 8a, the *query* (shown in dark shade within QR, white space corresponding to Xs for padding) matches a sub-sequence of the *reference* which is entirely stored in a single row of the array. We call this scenario a *full match*. On the other hand, in Fig. 8b, the *query* matches a sub-sequence of the *reference* which is stored in two consecutive rows of the array. We call this scenario a *fragmented match*. Fragmentation can happen at both ends of the *query*. For example, in Fig. 8b, the first portion of the *query* (shown in darker shade) matches the end of row j, while the rest (shown in lighter shade) matches the beginning of the next row, row j+1. MatchCtrl needs to address such fragmentation as GeNVoM lays out the character string representing the *reference* in each array *consecutively*.

Conventional TCAM can only detect full match. Handling fragmented match requires extra logic. By default, the TCAM array would select the longest sub-sequence *l* of the *reference* matching the input *query* if a full match is not the case, where *l* occupies an entire row. The darker-shade region in Fig. 8b corresponds to such *l*. As *l* may be aligned to either the beginning (Fig. 8b) or the end of the *query*, MatchCtrl has to additionally check the next or the previous row, respectively, for a match to the unmatched portion of the *query*. We call the first case a *fragmented tail match*; the second, a *fragmented head match*. In case of a fragmented match, search in the TCAM array takes two steps. As a fragmented match may also happen at TCAM array boundaries, each array's last row duplicates the first row of the next array in sequence.

Putting it All Together. Fig. 7 summarizes the 6 steps in mapping a *query* to the *reference*: First, FilterU retrieves a new *query* from the head of the input queue at step ①. In this case *seed*=7 (bases) with the corresponding 7-base *prefix* of the *query* underlined. Then, at step ②, FilterU locates the entry for the 7-base *prefix* of ACCCTGA in PMITIL, and extracts the corresponding PMIT address(es). Next, at step ③, FilterU retrieves TCAM array, column, and row numbers ($array_{\#}$, $col_{\#}$, and $row_{\#}$, for targeted search in MatchU) for the sub-sequences of the *reference* which match the *prefix* ACCCTGA, from the PMIT addresses collected at step ②. Finally, FilterU sends the *query* along with $array_{\#}$, $col_{\#}$, and $row_{\#}$ to MatchU over the search queue at step ④. At step ⑤, DispatchU initiates search in TCAM array $array_{\#}$, at $row_{\#}$ and $col_{\#}$, and collects the match outcome. At step ⑥, MatchU sends the match outcome to the output queue.

4 GENVOM: MICROSCOPIC VIEW

4.1 Search Space Pruning

In order to prune the search space, GeNVoM first locates sub-sequences of the *reference* matching the *seed*-long *prefix* of the *query* in FilterU (Section 3.3). *seed* represents a key

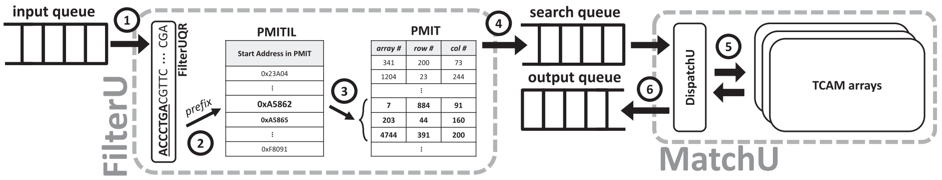


Fig. 7. Life-cycle of a *query* in GeNVoM.

GeNVoM design parameter which dictates not only the storage complexity, but also the degree of search space pruning, which in turn determines GeNVoM's throughput performance and energy efficiency.

PMITIL grows with 4^{seed}, therefore, the larger the *seed*, the higher becomes the storage complexity. However, a larger *seed* is more likely to result in a lower number of *prefix* matches in the PMI tables, and hence, a lower number of targeted searches in the MatchU. In either case, the *seed* value remains much less than the expected length of the *query*.

PMIT can have at most as many entries as the total number of *seed* long sub-sequences contained within the *reference*. This practically translates into the length of the *reference*, as a *prefix* can start from each base position of the *reference* onward. As PMIT is organized to keep multiple occurrences of the same *prefix* consecutively, each PMITIL entry just keeps the start address in the PMIT for the first occurrence. PMIT, on the other hand, has to keep the $<$TCAM array number, column number, row number$>$ tuple for each *prefix* match. If the *reference* is the human genome, PMIT would have approximately 3Giga entries. As we will detail in Section 5.2, 32 bits suffice to store each $<$TCAM array number, column number, row number$>$ tuple per PMIT entry; and 32 bits, each $<$PMIT start address$>$ per PMITIL entry.

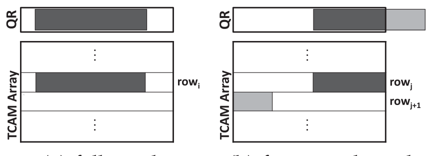
PMIT keeps the entries corresponding to the very same *prefix* always at consecutive addresses, and re-orders such entries further to have all entries pointing to the same TCAM array reside at consecutive addresses. GeNVoM processes multiple PMIT matches per *prefix* in this order. Under such re-ordering, communicating a list of PMIs and performing search in the array happen in a pipelined fashion. This masks communication latency and consequently, can improve throughput performance significantly.

4.2 Data Representation

Each input *query* and the *reference* itself represent character strings over the alphabet {A, G, C, T}. Conventional bioinformatics formats such as FASTA [53] encode each letter from such alphabets of bases by single-letter ASCII codes. However, TCAM arrays conduct the search at bit granularity. Therefore, GeNVoM needs to translate *base character* mismatches to *bit* mismatches. To this end, GeNVoM adopts an encoding which renders the very same number of mismatched bits for a mismatch between any two base characters. This would not be the case, if we encoded each base character in {A, G, C, T} by simply using 2 bits: a base-mismatch would sometimes cause a 2-bit mismatch (e.g., when comparing '01' to '10'); other times, a single-bit mismatch (e.g., when comparing '00' to '10'). GeNVoM's encoding instead uses 3 bits per base character, where any two 3-bit code-words differ by exactly 2 bits, such as {111, 100, 010, 001}. Thereby GeNVoM guarantees that exactly 2 bits would mismatch for any base character mismatch.

4.3 Similarity Search

Conventional TCAM arrays, including non-volatile variants such as [7], are designed to signal a (row-wise) match only if all bits within a row match with all bits in the query register. Therefore, under conventional operation semantics, even a single bit mismatch between a row and the *query*



(a) *full match* (b) *fragmented match*

Fig. 8. TCAM match scenarios.

renders a (row-wise) mismatch. This prevents a direct adaption of conventional TCAM for similarity matching, as a matching *query* may indeed have a few bases different from the corresponding sub-sequence of the *reference* (residing in a row) due to NGS or genomic variation induced noise (Section 3.1).

To resolve this discrepancy, GeNVoM deploys non-volatile TCAM arrays (based on [7]) capable of signaling a (row-wise) match in the presence of a few number of bit-wise mismatches. Specifically, we can tune these arrays to signal a row-wise match when less than a given number t of bits mismatch. We will refer to t as the *tolerance* – the number of acceptable base-wise mismatches, which represents an adjustable design parameter.

By construction, as t increases, distinguishing a (row-wise) match from a mismatch becomes more challenging [7]. GeNVoM's data encoding, which results in exactly 2-bit mismatches per base mismatch (Section 4.2), helps in this case: For instance, tolerating $t=1$ base mismatch translates into tolerating exactly $t\times 2 = 2$ bit mismatches in signaling a row-wise match. In other words, only if 2 or more bases, i.e., ≥ 4 bits – and not 3 bits – mismatch, a row-wise mismatch is the case. Differentiating a 2-bit mismatch from a 4-bit mismatch is easier than differentiating a 2-bit mismatch from a 3-bit mismatch.

In Section 6.2 we will characterize the resulting mapping accuracy, considering all sources of noise, including the impact of Process, Voltage, Temperature (PVT) variation and fragmented matches (Fig. 8). A fragmented match of a *query* may have $2 \times t$ base mismatches (as a result of two mapping steps, to cover both fragments), which can lead to rare cases of false matches.

4.4 Hierarchical Multi-Phase Search

Our focus so far was on the very basics of GeNVoM's mapping mechanism. We will next look into the mapping accuracy, specifically, under what circumstances GeNVoM may not be able to map a given *query* to the respective *reference*, which in fact was *similar enough*. In the following, we will refer to such cases as *missed* mappings. NGS imperfections (i.e., read errors) and genomic variations complicate mapping, and thereby can lead to misses.

As explained in Section 3.1, GeNVoM is designed to operate under all prevalent manifestations of read errors and genomic variations. To this end, GeNVoM employs multi-phase hierarchical mapping. Fig. 9 provides an overview. Each phase acts as a filtering layer for the subsequent phase, which in turn performs more complex mapping. *More complex mapping* entails re-attempting by considering *more complex manifestations* of read errors/genomic variations than the predecessor phase did. In this manner, each phase re-attempts mapping only for the subset of *query*s that the previous phase missed.

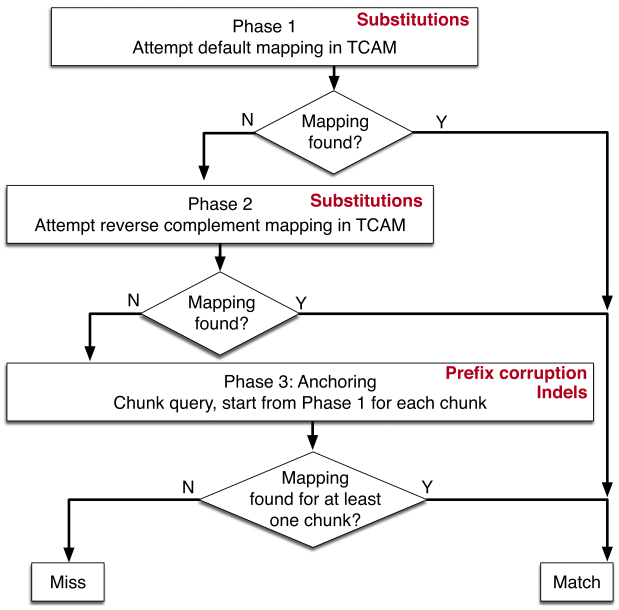

Phase 1
Substitutions
Attempt default mapping in TCAM
Mapping found?
N
Y
Phase 2
Substitutions
Attempt reverse complement mapping in TCAM
Mapping found?
N
Y
Phase 3: Anchoring
Prefix corruption
Indels
Chunk query, start from Phase 1 for each chunk
Mapping found for at least one chunk?
N
Y
Miss
Match

Fig. 9. GeNVoM's hierarchical multi-phase search flow.

If the mapping in a phase fails, MatchU raises the *Missed-Map* signal. GeNVoM in turn feeds *Missed-Map* back to FilterU, to trigger more complex mapping attempts in the subsequent phase(s).

Phase 1 attempts a basic mapping in the TCAM arrays, assuming that a reverse complement (Section 3.1) is not the case. In Phase 1, FilterU and MatchU closely follow the steps detailed in Section 3.3. GeNVoM TCAM search, by construction, can effectively identify similarity under base substitutions, which represent the common case per Fig. 3. Phase 1 can miss a mapping under three cases:

(i) *Reverse complement:* the *query* is a *read* sequenced in reverse order. The probability for this case, $P(i)$, is approximately 50%.

(ii) *Prefix corruption:* the *query*'s *seed*-long *prefix* (used for search space pruning in FilterU) has substitutions or indels. A corrupted *prefix* may lead to ill-addressed search requests, i.e., FilterU sending incorrect PMIs to MatchU. If the probability of having a corruption in a given base location is $P(loc)$, the probability for this case, $P(ii)$, becomes $1-(1-P(loc))^{seed}$. We can estimate $P(loc)$ by adding a typical read error rate of 0.1% [46] to an average genome variation rate of 0.1% [47], [48], [49]. Using this estimate, for a representative *seed* value of 15 (Section 5), $P(ii)$ barely reaches 3.0%.

(iii) *Indels:* the *query* contains indels, anywhere. The most common indels are short indels induced by genome variations. Let $P(indel)$ be the probability of a short indel, and len, the length of the *query*. Then, $P(iii) = 1-(1-P(indel))^{len}$ applies. While there is no consensus on $P(indel)$, 0.01% represents a conservative estimate [47], [54], which renders $P(iii) \approx 1.5\%$ for a typical *query* length of 150 [1].

Phase 2 handles missed mappings due to reverse complements. After getting *Missed-Map* from MatchU (at the end of Phase 1), FilterU immediately sends PMIs corresponding to the reverse complement of the *query* to MatchU. To accelerate processing, MatchU employs an extra register inside the Shift Logic, which keeps the reverse complement of the *query* in addition to the original. MatchU copies the reverse complement in this register at the time it gets the original *query* (during Phase 1). Therefore, upon receipt of *Missed-Map*, FilterU does not need to broadcast the reverse complement separately, but only the PMIs for the reverse complement (which FilterU simply extracts by consulting the PMI tables with the *seed*-long *prefix* of the reverse complement.)

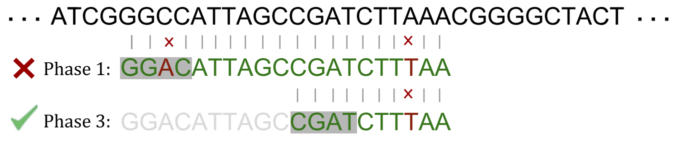

· · · ATCGGGCCATTAGCCGATCTTAAACGGGGCTACT · · ·
Phase 1: GGACATTAGCCGATCTTTAA
Phase 3: GGACATTAGCCGATCTTTAA

Fig. 10. An example of anchoring.

Phase 3 handles missed mappings due to *prefix* corruptions and indels, by adapting *anchoring* [52]. This phase processes all *query*s which Phase 2 was not able to map. Phase 3 first anchors the *query* in the middle to chunk the *query* uniformly into two. Then, each chunk separately goes through Phase 1, and if necessary, through Phase 2. Unless Phase 1 (or Phase 2, as need be) manages to map at least one of the two chunks to the *reference*, Phase 3 is considered to miss the mapping.

Fig. 10 depicts an example where Phase 3 maps a *read* which Phase 1 fails to map due to prefix corruption. The top row depicts the relevant portion of the *reference*. The second and third rows show the corresponding *read*, with pointers to the matching outcome at Phase 1 and 3, respectively. The alignment of the *read* w.r.t. the *reference* reflects the ideal alignment (which renders the most similar mapping). The shaded portion corresponds to the *prefix* (of length 4 in this case). The *read* and *prefix* lengths are not representative, but ease illustration. Phase 1 fails to identify this alignment due to the single base corruption (C→A) in the third base of the *prefix*. Phase 2 is of not much help either, as a reverse complement is not the case. Phase 3 comes to rescue, by chunking, i.e., *anchoring* the *read* in the middle, and attempting mapping for each half. The mapping of the first half still fails in this case, due to the very same corruption in the *prefix*. The *prefix* of the second half (i.e., CGAT), however, is not corrupted, and GeNVoM TCAM arrays can easily handle the single base mismatch in this half, which renders the correct alignment as a result – simply following the alignment dictated by the second half for the entire *read*.

Similar to this example, the evaluated GeNVoM design adapts 2-way chunking for anchoring, where multi-way (and not necessarily uniform) chunking can further help reduce the number of missed mappings. Anchoring essentially is a *divide and conquer* method. The key insight is that corruptions to lead to missed mappings are much less likely to occur in all of the shorter chunks simultaneously. In Section 6.2, we will demonstrate how anchoring can improve mapping accuracy significantly.

Phase 3 can miss a mapping under the very same two conditions as Phase 1 (namely *prefix* corruptions and indels anywhere; cases *(ii)* and *(iii)*), but only if these conditions apply to both of the chunks under *anchoring*. Under uniform two-way chunking, a typical *query* length of 150 bases renders a chunk length of 75, for which $P(ii)$ and $P(iii)$ become 3.0% and 1.0%, respectively. Hence, the probability to miss

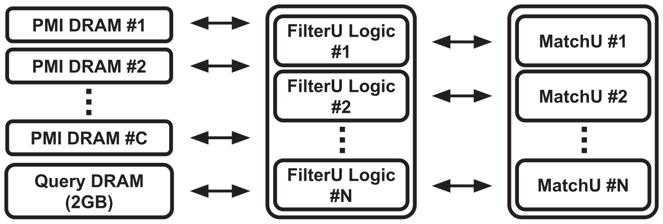


Fig. 11. Organization of a single GeNVoM card.

the mapping of a chunk would be approximately 4.0%. As Phase 3 can miss a mapping only by missing both chunks, the probability of missing a mapping in Phase 3 becomes approximately $(4.0\%)^2 = 0.16\%$.

By construction, anchoring can also help with more complex scenarios than indels or prefix corruptions, including long-indels, gaps, base duplications or inversions, as long as at least one chunk of the affected *query* is still "mapable" by GeNVoM. While anchoring reduces the miss probability significantly, Phase 3 may still miss mappings due to rare complex variations. To map such problematic *query*s, GeNVoM can be paired with sophisticated software algorithms.

5 Evaluation Setup

5.1 System-Level Characterization

Without loss of generality, all components of the evaluated GeNVoM design reside in a single card attached to the PCIe bus. The host loads the *reference*, PMIT, and PMITIL tables to the GeNVoM card before starting mapping by sending GeNVoM the range of addresses of *read*s inside the main memory and a dedicated memory space to write back the results, along with the *read* length. To explore GeNVoM's design space, we experiment with different numbers (N) of FilterUs and MatchUs under a fixed problem size. Fig. 11 provides the overview. In this case, each MatchU stores $1/N$ of the *reference* and the FilterU paired with it only covers the PMIs for that portion of the *reference*.

This translates into banking PMIT to assign the FilterU only one bank covering the *reference* portion in the paired MatchU. As the problem size (which is dictated by the *reference* length) is fixed, such distributed GeNVoM configurations don't result in larger TCAM arrays or PMIT. However, each PMIT bank still needs a separate index locator, i.e., PMITL. As discussed in Section 3.3, PMITIL capacity depends on the *seed* only, not the size of PMIT. Therefore, N GeNVoM pipelines (with each processing only $1/N$ of the *reference*) still require N full-size PMITILs (of 4^{seed} entries).

In the rest of the evaluation, we characterize each GeNVoM configuration by four parameters: M, C, L, and N. A total on-card DRAM space of MGB is required to store all PMIT and PMITIL. This MGB of memory feeds the GeNVoM pipeline via C channels. N represents the number of FilterU+MatchU pairs, and L, the *seed* value.

GeNVoM relies on a GPU kernel (based on [55]) to re-format the match outcome of mapped *query*s (i.e., to find different SAM format fields such as CIGAR and MAPQ). The kernel processes mapped *query*s in a pipelined manner, periodically. This significantly reduces the throughput overhead of the GPU kernel, and we quantify its energy-efficiency implications by real measurements (similar to Section 5.5).

TABLE 1
PMITIL Table Parameters as a Function of *Seed*

seed	10	11	12	13	14	15
Size (GB)	0.004	0.017	0.067	0.268	1.074	4.295
Avg. # searches	21.4K	12.8K	8.8K	6.5K	4.9K	3.8K

To quantify throughput and energy-efficiency, we use Ramulator [56]. The default Ramulator can only model accesses to the DRAM chips, so we implement intra-card interactions during search, control and network related operations. We assume LPDDR4-4266 DRAM to store PMIT, PMITIL, and the *query* dataset. We use DRAMPower [57] to estimate the power consumption, which doesn't support LPDDR4. Therefore, we conservatively report DRAM power based on DRAMPower's LPDDR3 model.

5.2 PMI Table Generation

As explained in Section 4.1, PMIT keeps an entry for each possible *seed*-long *prefix* contained within the *reference*. Therefore, PMIT capacity becomes practically independent of the *seed* for feasible *seed* values. A tight-enough upper bound for PMIT capacity for the human genome used as the *reference* for evaluation (Section 5.6) is approximately 11.4GB, independent of the *seed*.

On the other hand, PMITIL contains 4^{seed} entries. We evaluate GeNVoM considering different *seed* values. Table 1 captures PMITIL capacity and average number of locations to search for practical *seed* values ranging from 10 to 15. PMIT size and $N\times$PMITIL size together, M, determine the DRAM space requirement of the evaluated GeNVoM implementation.

5.3 Circuit-Level Characterization

The evaluated GeNVoM implementation uses Phase Change Memory (PCM) as the resistive memory technology for TCAM arrays, which features a relatively high R_{high} to R_{low} ratio: 11.5 on average [58]. In a TCAM array, a bitwise match renders R_{high}; a bitwise mismatch, R_{low} on the match-line. A higher R_{high} to R_{low} ratio therefore eases sensing, and enables arrays with longer rows. We experiment with 1K$\times$1Kbit TCAM arrays.

We synthesize logic circuits by Synopsys Design Compiler vH2013.12 using the FreePDK45 library [59]. To match the technology of our baselines for comparison (Section 5.5), we scale the outcome from 45nm to 28nm using ITRS projections [60]. GeNVoM's logic operates at 1GHz. A single search operation takes 0.9ns to complete, while consuming 0.1nJ of energy. We use ORION2.0 [61] to model the network. The H-tree network connecting TCAM arrays operates at 1GHz, while each hop (1 router + link) consumes 3.83mW.

5.4 Similarity Matching Specification

Without loss of generality, GeNVoM uses IBM's fabricated TCAM arrays employing 2T-2R PCM-based cell [7], which feature tunable matching, via a dedicated configuration row to set the acceptable number of mismatching bits to render a row-wise match. We simulate this matching mechanism in HSPICE v2015.06 using the FreePDK45 [59] library, to measure performance and *accuracy under PVT variation.*

Tunable Matching. In a TCAM array, a bitwise match renders a high resistance; a bitwise mismatch, a low resistance on the match-line. In other words, each cell within each row contributes to the effective resistance connected to the match-line by R_{high} (R_{low}) on a match (mismatch). Under PVT variations, R_{high} and R_{low} values may deviate from their nominal values. We conduct a Monte Carlo analysis using the (variation-afflicted) high and low resistance distributions from [58], extracted from measured data: $\mu(R_{high}) = 243.8K\Omega$, $\sigma(R_{high}) = 50.9K\Omega$, $\mu(R_{low}) = 21.2K\Omega$, and $\sigma(R_{low}) = 2.5K\Omega$. μ and σ represent the mean and the standard deviation. Considering a row size of 1Kbits, we find the effective resistance of TCAM array's match-line for 1M sample scenarios, each corresponding to a different number of base mismatches. Using the resulting distribution, and capping the maximum number of base mismatches that are permitted to pass as a match (i.e., the *tolerance* as explained in Section 4.3), we configure each TCAM array in a variation-aware manner, in order to prevent false negatives, i.e., signaling a mismatch for a similar-enough *query*.

Matching Accuracy. Under PVT variations, potential shifts in R_{high} and R_{low} levels may trigger a (row-wide) match in case of an actual (row-wide) mismatch. For each such case, the number of base mismatches remains higher than the preset *tolerance* value. In the following, we will refer to this difference in the number of base mismatches with respect to the *tolerance* as *overshoot*. These cases are not necessarily errors, and rather translate into a *query* of less similarity than expected being matched to a sub-sequence of the *reference*. Therefore, as long as the overshoot (in terms of base mismatches) with respect to the anticipated *tolerance* remains bounded, each such case can pass as a *less similar match* (which in fact can be an actual match where the input *query* was significantly corrupted). Monte Carlo analysis shows that *under PVT variations*, for different representative *tolerance* values used in Section 6, overshoot is usually less than 3, with probability of an overshoot of size 3 or larger barely reaching 0.05%. We quantify the impact of this on GeNVoM's overall accuracy in Section 6.2.

5.5 Baselines for Comparison

As comparison baselines, we pick a highly optimized GPU implementation of the popular BWA algorithm, SOAP3-dp [15], and a very efficient hardware accelerator for short *read* alignment, GenAx [6]. A pure software-based implementation of GeNVoM is orders of magnitude slower than SOAP3-dp. We evaluate the throughput performance and power consumption of SOAP3-dp on an NVIDIA Tesla K40 GPU. We measure the power consumption of the GPU using NVIDIA-SMI (System Management Interface) command. We use the same *reference* and *query* dataset (Section 5.6) as GeNVoM as inputs. We compare GeNVoM against two different configurations of SOAP3-dp: The first one, SOAP$_{SUB}$, only handles substitutions; the second one, SOAP$_{ALL}$, captures prevalent forms of read errors and genomic variations.

5.6 Input Dataset

We use a real human genome, `g1k_v37`, from the 1000 genomes project [62] as the *reference* genome; and 20 million 100-base long real *reads* from NA12878 [63] as a *query* dataset. For mapping accuracy analysis, we further generate 20 million more *query*s using 100-base long randomly picked sub-sequences from this *reference*, which we corrupt considering a read error probability of 0.1% (to mimic modern Illumina platforms), a single substitution[1] probability of 0.09%, and a short indel probability of 0.009%. For a fair comparison (not to favor GeNVoM) we choose the number of *query*s to have the *reference* + *query*s fit into the main memory of the GPU, such that the GPU does not suffer from extra energy-hungry data communication. We also limit the evaluation to a single GeNVoM card to keep the resource utilization comparable to the baselines.

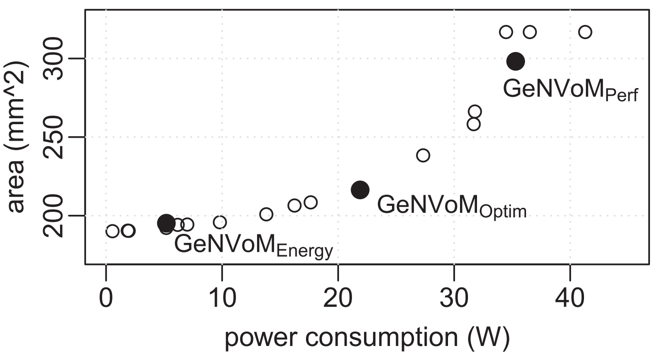


Fig. 12. Evaluated GeNVoM configurations.

5.7 Design Space Exploration

We sweep M, C, L, N (Section 5.1) to extract the Pareto fronts, and to identify the highest throughput ($GeNVoM_{Perf}$); highest energy-efficiency ($GeNVoM_{Energy}$); and highest throughput and energy-efficiency ($GeNVoM_{Optim}$) GeNVoM configurations. M ranges from 16GB to 128GB; L, from 10 to 15. For each L, we find the maximum N where PMIT size and $N\times$PMITIL size fits in the given M budget. Besides, we cap N at 512, to limit C, number of LPDDR4-4266 channels feeding PMIs to FilterUs, at 64. For reference, Fig. 12 depicts the power versus area trade-off for each evaluated GeNVoM configuration covering these ranges. The three marked points reflect the three Pareto designs, explored in Section 6.

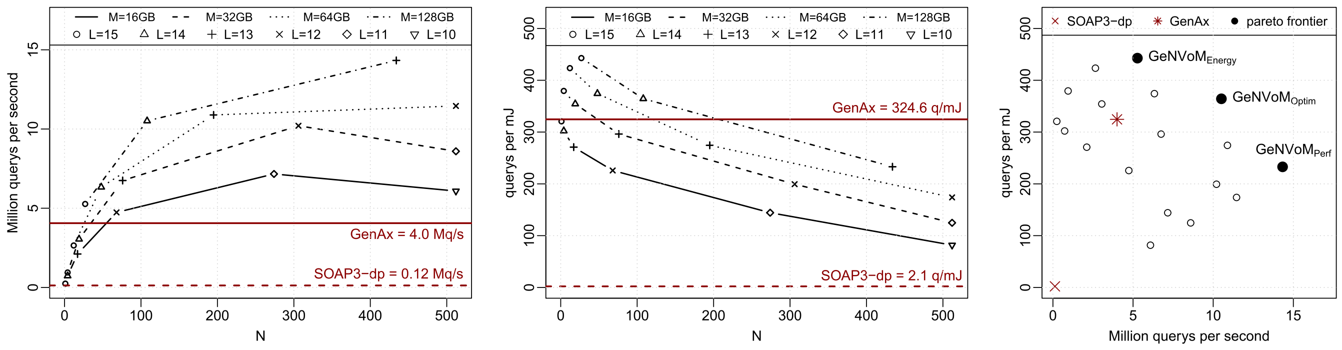
6 Evaluation

6.1 Throughput Performance and Energy

Larger *seed* values (i.e., L) prune the search space more, resulting in a progressively lower number of search operations in processing each *query*. For example, increasing L from 10 to 15 decreases average number of search operations per *query* from 21.4K to 3.8K. On the other hand, lower L leads to smaller PMITIL tables. As the memory footprint increases with N $\times$ PMITIL size, for a given memory budget M, lower L makes higher N possible. And higher N translates into more FilterU + MatchU pairs (each processing a portion of the *reference* in parallel). We will next look into this trade-off.

Fig. 13a captures the throughput. Y-axis shows the number of *query*s mapped per second; X-axis, the number of FilterU + MatchU pairs, N. The two horizontal lines correspond to the throughput of the two baselines, SOAP3-dp and GenAx, respectively. Each trendline corresponds to

1. Single-nucleotide polymorphism, SNP, the dominant type of substitutions induced by genomic variations.



(a) Throughput Performance (b) Energy (c) Trade-off

Fig. 13. Throughput performance and energy consumption.

a different M budget; each point on each trendline, to a different L. We don't include L values which result in $N > 512$ (Section 5.7).

According to Fig. 13a, for each M, starting from L=15 (on the left), decreasing L generally makes higher N possible, and consequently increases throughput. However, lower L also prune the search space less, resulting in a progressively higher number of search operations in processing each *query*. For M=32MB (16GB), at L values lower than 12 (11), this effect starts to become dominant, and wipes off the throughput benefits of the higher N (i.e., more FilterU + MatchU pairs operating simultaneously). Besides, we also observe how larger M budgets improve the overall throughput. While most of the GeNVoM configurations render a higher throughput than the baselines (i.e., remain above both baseline lines), the fastest design, $GeNVoM_{Perf}$, is 114.6× (3.67×) faster than SOAP3-dp (GenAx) for $M = 128GB$, $L = 13$, $C = 55$, and $N = 434$, consuming 35.3Watts on average and 298.3mm^2.

Fig. 13b captures energy-efficiency. Y-axis shows the number of *querys* mapped per mJoule of energy (q/mJ); X-axis, the number of FilterU + MatchU pairs, N. The two horizontal lines correspond to the q/mJ of the two baselines. Similar to Fig. 13a, each trendline corresponds to a different M budget; each point on each trendline, to a different L.

According to Fig. 13b, energy efficiency degrades with decreasing L. The reason is two-fold: Lower L increase the number of search operations per *query* due to less effective search space pruning. At the same time, lower L make higher values of N possible, which increases the power consumption. Still, all (many) GeNVoM configurations are more energy-efficient than SOAP3-dp (GenAx). The most energy efficient design, $GeNVoM_{Energy}$, with 442.3 q/mJ, corresponds to $M = 128GB$, $L = 15$, $C = 4$, and $N = 27$. $GeNVoM_{Energy}$ improves energy-efficiency of SOAP3-dp (GenAx) by 210.9× (1.36×), while consuming 5.2Watts on average and 195.3mm^2.

Putting it all together, Fig. 13c depicts the trade-off space of throughput (x-axis) versus energy-efficiency (y-axis) for the evaluated GeNVoM configurations and the baselines. Points closer to the top-right corner are faster and more energy-efficient. All GeNVoM configurations outperform SOAP3-dp in both throughput and energy efficiency, while a few are *both* faster and more energy-efficient than GenAx. At the Pareto-frontier reside $GeNVoM_{Perf}$, $GeNVoM_{Energy}$, and also $GeNVoM_{Optim}$ which represents a sweet spot for both performance and energy-efficiency. $GeNVoM_{Optim}$ corresponds to $M = 128GB$, $L = 14$, $C = 14$, and $N = 108$, consuming 21.9Watts on average and 216.5mm^2. $GeNVoM_{Optim}$ can map 10.5M *querys* per second (84.8× faster than SOAP3-dp; 2.63×, than GenAx) and 364.1 *querys* per mJ (173.5× better than SOAP3-dp; 1.12×, than GenAx). All three Pareto-frontier configurations correspond to $M = 128GB$, which indicates that GeNVoM benefits from larger memory space for both throughput and energy-efficiency.

Without loss of generality, for $L = 15$, GeNVoM maps 45.4% of the *querys* in Phase 1; 42.1% in Phase 2 (reverse complement); and 8.4%, in Phase 3. Specifically, in Phase 3, GeNVoM maps 3.0% of the *querys* after anchoring the first half; 2.6% after anchoring the second half; 1.8% after anchoring reverse complement of the first half; and 1.0% after anchoring reverse complement of the second half, respectively. This renders a mapping rate (i.e., the share of successfully mapped *querys* over all) of around 96.0% for GeNVoM, while SOAP3-dp can map 97.4% of the *querys*.

6.2 Mapping Accuracy

Since we were not able to run experiments using GenAx, we compare the accuracy of GeNVoM to SOAP3-dp only. As SOAP3-dp outperforms the accuracy of BWA-SW [15], which has a very similar accuracy to GenAx, SOAP3-dp is expected to be more accurate than GenAx.

To compare the mapping accuracy of GeNVoM to SOAP3-dp, we use a simulated input dataset with known expected matching indices. We differentiate between two cases: (1) the *query* is aligned to a wrong portion of the *reference*; or (2) the *query* is not aligned to any portion of the *reference*. Table 2 shows the corresponding rate of occurrence for (1) as the *Misalignment rate*; for (2), as the *Miss rate*, considering different configurations.

Misaligned *querys* are still mapped to a portion of the *reference*, which may be similar enough. Therefore, *misalignment rate* does not necessarily correspond to an error rate. The numbers in Table 2 reflect all problematic cases for GeNVoM due to prefix corruptions, indel mishandling or false positives.

We further observe that both *misalignment rate* and *miss rate* slightly increase with larger L values (as we move right from GeNVoM$_{Energy}$ to GeNVoM$_{Perf}$), due to higher prefix corruption probability. However, the overall accuracy of different GeNVoM configurations considering different L values stay in a similar range. Generally, GeNVoM fails to align only around 0.03% more *querys* than SOAP3-dp, while misaligning around 1.85% more.

TABLE 2
Mapping Accuracy of GeNVoM w.r.t. SOAP

	GeNVoM$_{Energy}$	GeNVoM$_{Optim}$	GeNVoM$_{Perf}$	SOAP3-dp
Misalign. Rate	2.94%	2.97%	2.99%	1.12%
Miss Rate	0.03%	0.03%	0.04%	0.01%
Total	2.97%	3.0%	3.03%	1.13%

Next, we evaluate the effectiveness of GeNVoM's anchoring phase (Phase 3 from Section 4.4), in improving the accuracy of mapping over the first two phases. Without loss of generality, we use GeNVoM$_{Energy}$ as a case study, which fails to map 2.88% of the *query*s after the first two phases. However, Phase 3 manages to map 98.6% of the *query*s that the first two phases missed (due to indels and prefix corruption), while aligning 91.2% of them to the correct index of the *reference*. This analysis demonstrates how effective multi-phase search is in improving the mapping accuracy when dealing with *query*s failed to be mapped by the first two phases. Besides, out of the 2.99% misaligned reads, 0.01% are misaligned due to false positives (FP) of Phase 3, 0.25% due to FPs of prefix corruptions, 0.8% due to FPs of partial matches, and 1.93% due to FPs induced by *PVT variations*. Hence, replacing PCM with other technologies featuring a higher R_{high}/R_{low} ratio can significantly reduce the number of misaligned reads, by up to 64.5%.

SOAP3-dp lets users trade accuracy for higher throughput. In this mode, SOAP3-dp turns off its complex, dynamic programming based mapping mechanism, and only considers substitutions. The user can set the accuracy level by tuning the number of acceptable mismatches, which is similar to the *tolerance* level of GeNVoM. A lower number of acceptable mismatches implies a lower *tolerance*. Lower *tolerance* leads to lower mapping accuracy – as chances of missing the mapping of even similar-enough *query*s with a few corruptions increases. At the same time, lower *tolerance* improves SOAP3-dp's throughput – as there is no need for sophisticated processing to cover complex corruptions. On the other hand, *tolerance* does not affect GeNVoM's throughput noticeably, since neither FilterU's nor MatchU's performance depend on it. More specifically, GeNVoM can tune *tolerance* without compromising search throughput.

Fig. 14 depicts how SOAP3-dp's mapping (in)accuracy (left y-axis) changes with *tolerance* in terms of the number of acceptable mismatches (x-axis). The left y-axis corresponds to the *Total* row of Table 2. The bar labeled by *DP* captures the default for SOAP3-dp, which relies on complex, dynamic programming based mapping. The trendline captures GeNVoM$_{Perf}$'s speed-up w.r.t. SOAP3-dp (right y-axis). We observe that, for smaller *tolerance* values, SOAP3-dp becomes less accurate, and at the same time faster, which decreases GeNVoM's speed-up over SOAP3-dp. The lowest inaccuracy of SOAP3-dp, 2.88% for a *tolerance* of 0, is close to the inaccuracy of GeNVoM$_{Perf}$. However, being 21.5% faster than the default SOAP3-dp, this point is still 93.4× slower than GeNVoM$_{Perf}$. Hence, GeNVoM outperforms even an iso-accuracy baseline by approx. two orders of magnitude.

A Note on Acceptability. In a typical NGS *query* dataset, all *query*s, if concatenated back to back, would be at least $50\times$ longer than the *reference* genome [64]. Therefore, even if we miss the mapping of a few percent of the *query*s (due

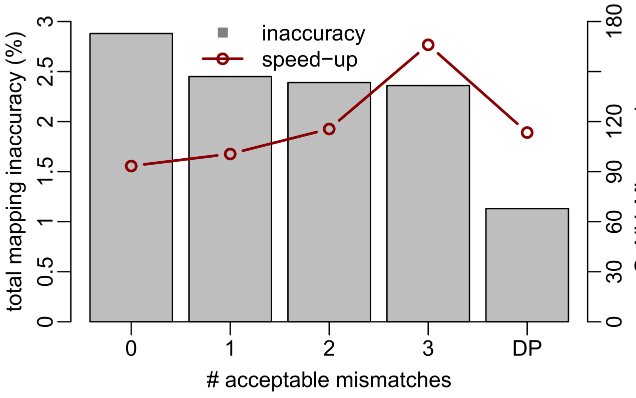


Fig. 14. *tolerance* versus accuracy and peformance.

to different read errors/genome variations), we still have plenty of *query*s to cover such missed regions of the *reference*. The average number of *query*s covering any given base in the *reference* genome is called the *depth*. Only missing all *query*s covering a base of the *reference* would be problematic.

What is the probability for GeNVoM to miss all *query*s covering a given base of the *reference*? Let the number of *query*s covering a given base be Q, and let $min(Q)$ denote its minimum. Q follows a Poisson distribution [65]. For a representative *depth* of 50, the probability of having a base covered by less than 10 *query*s is 5.2×10^{-12}, practically negligible. Therefore, we can assume that all bases are covered at least by 10 *query*s, i.e., that $min(Q) = 10$. The worst-case probability of GeNVoM missing a mapping, P_{miss} is around 3.03% (Table 2). Hence, the probability of GeNVoM missing all *query*s covering a base of the *reference* becomes ${P_{miss}}^{min(Q)} = 6.5 \times 10^{-16}$, which is practically negligible.

7 Discussion & Conclusion

This paper proposes GeNVoM, a novel *read mapping* accelerator which unlocks the throughput potential of non-volatile TCAM. GeNVoM results in up to 114.6× (3.67×) higher throughput while consuming up to 210.9× (1.36×) less energy when compared to a highly-optimized GPU (ASIC) implementation.

Currently, short (i.e., 100-200 base long) *read*s from modern Illumina NGS platforms [1] constitute more than 90% of all *read*s in the world. This dominance is unlikely to quickly change in the near future due to the progressively dropping sequencing cost of short *read* technologies, rendering them significantly more cost-efficient than the long *read* counterparts such as PacBio [66] or Oxford Nanopore [67] (where *read* lengths can exceed tens of thousands of bases). The key benefit of long *read* sequencing technologies comes from the capability of directly extracting long-range information, and not necessarily from higher accuracy. That said, many emerging recent technologies such as 10xGENOMICS [68] can obtain long-range information from short *read*s. Although it is very hard to predict the future exactly, considering practical facts such as market share and market caps, we believe that short *read* platforms will remain prevalent at least in the near future. So, GeNVoM is designed for short *read mapping*.

Short *read mapping* and long *read mapping* represent two fundamentally different problems. This is because of the difference in NGS and genomic variation induced noise manifestations. Long *read*s are subject to more frequent and complex corruptions, which gives rise to very different algorithms for the mapping problem than short *read*s. For example, perturbation by a significant number of indels is not uncommon [69]. A very efficient hardware accelerator for long *read mapping* has also been proposed recently [70]. As such accelerators are highly optimized for long *read*s (which suffer from more complex noise manifestation), they are by construction sub-optimal for short *read mapping* (for which substitutions are dominant per Fig. 3).

GeNVoM features a rich design space. 3D stacking is an option, for example, to enable even more parallelism subject to a stringent thermal budget, where the scale of the problem demands careful optimization for data communication between the memory modules and logic embedded in/near memory, which we reserve for future work.

References

[1] Illumina Sequencing by Synthesis (SBS) Technology. Accessed: Dec. 1, 2019. [Online]. Available: https://www.illumina.com/technology/next-generation-sequencing/sequencing-technology.html

[2] Z. D. Stephens *et al.*, "Big data: Astronomical or genomical?," *PLoS Biol.*, vol. 13, no. 7, Jul. 2015, Art. no. e1002195.

[3] S. Aluru and N. Jammula, "A review of hardware acceleration for computational genomics," *IEEE Des. Test*, vol. 31, no. 1, pp. 19–30, Feb. 2014.

[4] P. Klus *et al.*, "BarraCUDA - A fast short read sequence aligner using graphics processing units," *BMC Res. Notes*, vol. 5, no. 1, Jan. 2012, Art. no. 27.

[5] Y. Chen, B. Schmidt, and D. L. Maskell, "A hybrid short read mapping accelerator," *BMC Bioinf.*, vol. 14, no. 1, 2013, Art. no. 67.

[6] D. Fuijiki *et al.*, "GenAx: A Genome Sequencing Accelerator," *Proc. ACM/IEEE Int. Symp. Comput. Architecture*, 2018, pp. 69–82.

[7] J. Li *et al.*, "1Mb 0.41 μm2 2T-2R cell nonvolatile TCAM with two-bit encoding and clocked self-referenced sensing," in *Proc. Symp. VLSI Circuits*, 2013, pp. C104–C105.

[8] R. Li, Y. Li, K. Kristiansen, and J. Wang, "SOAP: Short oligonucleotide alignment program," *Bioinformatics*, vol. 24, no. 5, pp. 713–714, 2008.

[9] H. Li, J. Ruan, and R. Durbin, "Mapping short DNA sequencing reads and calling variants using mapping quality scores," *Genome Res.*, vol. 18, no. 11, pp. 1851–1858, 2008.

[10] R. Li *et al.*, "SOAP2: An improved ultrafast tool for short read alignment," *Bioinformatics*, vol. 25, no. 15, pp. 1966–1967, 2009.

[11] B. Langmead, C. Trapnell, M. Pop, and S. L. Salzberg, "Ultrafast and memory-efficient alignment of short DNA sequences to the human genome," *Genome Biol.*, vol. 10, no. 3, 2009, Art. no. R25.

[12] B. Langmead and S. L. Salzberg, "Fast gapped-read alignment with bowtie 2," *Nat. Methods*, vol. 9, no. 4, pp. 357–359, 2012.

[13] H. Li and R. Durbin, "Fast and accurate short read alignment with burrows–wheeler transform," *Bioinformatics*, vol. 25, no. 14, pp. 1754–1760, 2009.

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

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

[16] Z. Wang *et al.*, "A 2.46m reads/s genome sequencing accelerator using a 625 processing-element array," in *Proc. IEEE Custom Integrated Circuits Conf.*, 2020, pp. 1–4.

[17] J. Cong, Z. Fang, M. Gill, F. Javadi, and G. Reinman, "AIM: Accelerating computational genomics through scalable and noninvasive accelerator-interposed memory," in *Proc. Int. Symp. Memory Syst.*, 2017, pp. 3–14.

[18] W. Vanderbauwhede and K. Benkrid, *High-Performance Computing Using FPGAs*. Berlin, Germany: Springer, 2013.

[19] A. Madhavan, T. Sherwood, and D. Strukov, "Race logic: A hardware acceleration for dynamic programming algorithms," in *Proc. ACM/IEEE Int. Symp. Comput. Architecture*, 2014, pp. 517–528.

[20] Y. Wu *et al.*, "21.1 a fully integrated genetic variant discovery SOC for next-generation sequencing," in *Proc. IEEE Int. Solid- State Circuits Conf.*, 2020, pp. 322–324.

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

[22] C. Bo, V. Dang, E. Sadredini, and K. Skadron, "Searching for potential gRNA off-target sites for CRISPR/Cas9 using automata processing across different platforms," in *Proc. IEEE Int. Symp. High Perform. Comput. Architecture .*, 2018, pp. 737–748.

[23] L. Wu *et al.*, "FPGA accelerated INDEL realignment in the cloud," in *Proc. IEEE Int. Symp. High Perform. Comput. Architecture*, 2019, pp. 277–290.

[24] J. S. Kim *et al.*, "GRIM-Filter: Fast seed location filtering in DNA read mapping using processing-in-memory technologies," *BMC Genomic.*, vol. 19, no. 2, 2018, Art. no. 89.

[25] M. Alser, H. Hassan, H. Xin, O. Ergin, O. Mutlu, and C. Alkan, "GateKeeper: A new hardware architecture for accelerating pre-alignment in DNA short read mapping," *Bioinformatics*, vol. 33, no. 21, pp. 3355–3363, 2017.

[26] H. Xin *et al.*, "Shifted hamming distance: A fast and accurate SIMD-friendly filter to accelerate alignment verification in read mapping," *Bioinformatics*, vol. 31, no. 10, pp. 1553–1560, 2015.

[27] R. Kaplan, L. Yavits, and R. Ginosar, "RASSA: Resistive pre-alignment accelerator for approximate DNA long read mapping," *IEEE Micro*, vol. 39, no. 4, pp. 1–1, Jul./Aug. 2018.

[28] Q. Guo, X. Guo, Y. Bai, and E. Ipek, "A resistive TCAM accelerator for data-intensive computing," *Proc. 44th Annu. IEEE Int. Symp. Microarchitecture*, 2011, pp. 339–350.

[29] Q. Guo, X. Guo, R. Patel, E. Ipek, and E. G. Friedman, "AC-DIMM: Associative computing with STT-MRAM," in *Proc. ACM/IEEE Int. Symp. Comput. Architecture*, 2013, pp. 189–200.

[30] L. Yavits, S. Kvatinsky, A. Morad, and R. Ginosar, "Resistive associative processor," *IEEE Comput. Architecture Lett.*, vol. 14, no. 2, pp. 148–151, Jul.–Dec. 2015.

[31] R. Kaplan, L. Yavits, R. Ginosar, and U. Weiser, "A resistive cam processing-in-storage architecture for DNA sequence alignment," *IEEE Micro*, vol. 37, no. 4, pp. 20–28, Aug. 2017.

[32] M. Imani, D. Peroni, A. Rahimi, and T. Rosing, "Resistive CAM acceleration for tunable approximate computing," *IEEE Trans. Emerg. Top. Comput.*, vol. 7, no. 2, pp. 271–280, Apr.–Jun. 2019.

[33] M. Imani, A. Rahimi, D. Kong, T. Rosing, and J. M. Rabaey, "Exploring hyperdimensional associative memory," in *Proc. IEEE Int. Symp. High Perform. Comput. Architecture*, 2017, pp. 445–456.

[34] B. Yan *et al.*, "A high-speed robust NVM-TCAM design using body bias feedback," in *Proc. 25th Ed. Great Lakes Symp.*, 2015, pp. 69–74.

[35] 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 *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, 2016, pp. 1–7.

[36] L. Yuan and D.-S. Huang, "A network-guided association mapping approach from DNA methylation to disease," *Sci. Rep.*, vol. 9, no. 1, pp. 1–16, 2019.

[37] B. Li, Z.-T. Fan, X.-L. Zhang, and D.-S. Huang, "Robust dimensionality reduction via feature space to feature space distance metric learning," *Neural Netw.*, vol. 112, pp. 1–14, 2019.

[38] Z. Shen, W. Bao, and D.-S. Huang, "Recurrent neural network for predicting transcription factor binding sites," *Sci. Rep.*, vol. 8, no. 1, pp. 1–10, 2018.

[39] Y. Yang *et al.*, "Machine learning for classifying tuberculosis drug-resistance from dna sequencing data," *Bioinformatics*, vol. 34, no. 10, pp. 1666–1671, 2018.

[40] W. Bao *et al.*, "IMKpse: Identification of protein malonylation sites by the key features into general pseAAC," *IEEE Access*, vol. 7, pp. 54073–54083, 2019.

[41] W. Bao, B. Yang, D. Li, Z. Li, Y. Zhou, and R. Bao, "CMSENN: Computational modification sites with ensemble neural network," *Chemometrics Intell. Lab. Syst.*, vol. 185, pp. 65–72, 2019.

[42] W. Bao *et al.*, "IMKpse: Identification of protein malonylation sites by the key features into general pseAAC," *IEEE Access*, vol. 7, pp. 54073–54083, 2019.

[43] W. Bao, D.-S. Huang, and Y.-H. Chen, "MSIT: Malonylation sites identification tree," *Curr. Bioinf.*, vol. 15, no. 1, pp. 59–67, 2020.

[44] W. Bao, B. Yang, Z. Li, and Y. Zhou, "LAIPT: Lysine acetylation site identification with polynomial tree," *Int. J. Mol. Sci.*, vol. 20, no. 1, 2019, Art. no. 113.

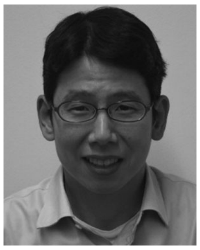
[45] W. Bao *et al.*, "Mutli-features prediction of protein translational modification sites," *IEEE/ACM Trans. Comput. Biol. Bioinf.*, vol. 15, no. 5, pp. 1453–1460, Sep. 2018.
[46] M. Schirmer, R. D'Amore, U. Z. Ijaz, N. Hall, and C. Quince, "Illumina error profiles: Resolving fine-scale variation in metagenomic sequencing data," *BMC Bioinf.*, vol. 17, no. 1, 2016, Art. no. 125.
[47] J. M. Mullaney, R. E. Mills, W. S. Pittard, and S. E. Devine, "Small insertions and deletions (INDELs) in human genomes," *Hum. Mol. Genet.*, vol. 19, no. R2, pp. R131–R136, 2010.
[48] S.-M. Ahn *et al.*, "The first Korean genome sequence and analysis: Full genome sequencing for a socio-ethnic group," *Genome Res.*, vol. 19, no. 9, pp. 1622–1629, 2009.
[49] J. Wala *et al.*, "Genome-wide detection of structural variants and indels by local assembly," 2017, *bioRxiv 105080*.
[50] R. P. Abo *et al.*, "BreaKmer: Detection of structural variation in targeted massively parallel sequencing data using kmers," *Nucleic Acids Res.*, vol. 43, no. 3, pp. e19–e19, 2014.
[51] J. Leskovec, A. Rajaraman, and J. D. Ullman, *Mining of Massive Datasets*. Cambridge, U.K.: Cambridge Univ. Press, 2014.
[52] B. Langmead, C. Trapnell, M. Pop, and S. L. Salzberg, "Ultrafast and memory-efficient alignment of short DNA sequences to the human genome," *Genome Biol.*, vol. 10, no. 3, 2009, Art. no. R25.
[53] D. J. Lipman and W. R. Pearson, "Rapid and sensitive protein similarity searches," *Science*, vol. 227, no. 4693, pp. 1435–1441, 1985.
[54] J. Wang *et al.*, "The diploid genome sequence of an asian individual," *Nature*, vol. 456, no. 7218, pp. 60–65, 2008.
[55] NVBIO Smith-Waterman. Accessed: Dec. 1, 2019. [Online]. Available: https://developer.nvidia.com/nvbio/
[56] Y. Kim, W. Yang, and O. Mutlu, "Ramulator: A fast and extensible DRAM simulator," *Comput. Architecture Lett.*, vol. 15, no. 1, pp. 45–49, 2016.
[57] K. Chandrasekar, C. Weis, Y. Li, B. Akesson, N. Wehn, and K. Goossens, "Drampower: Open-source dram power & energy estimation tool," 2012. [Online]. Available: http://www.drampower.info
[58] H. Y. Cheng *et al.*, "Novel fast-switching and high-data retention phase-change memory based on new Ga-Sb-Ge material," in *Proc. IEEE Int. Electron Devices Meeting*, 2015, pp. 3.5.1–3.5.4.
[59] NCSU-EDA, "FreePDK45." Accessed: Dec. 1, 2019. [Online]. Available: https://www.eda.ncsu.edu/wiki/FreePDK45:Contents
[60] B. Hoefflinger, "ITRS: The international technology roadmap for semiconductors," in *Chips*, Berlin, Heidelberg: Springer, 2020, pp. 161–174.
[61] A. B. Kahng, B. Li, L.-S. Peh, and K. Samadi, "Orion 2.0: A fast and accurate NOC power and area model for early-stage design space exploration," in *Proc. Des., Automat. Test Eur.*, 2009, pp. 423–428.
[62] 1000 Genomes Project. Accessed: Dec. 1, 2019. [Online]. Available: ftp://ftp-trace.ncbi.nih.gov/1000genomes/ftp/technical/reference/
[63] Na12878. Accessed: Dec. 1, 2019. [Online]. Available: ftp://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/data/NA12878/
[64] S. S. Ajay, S. C. Parker, H. O. Abaan, K. V. F. Fajardo, and E. H. Margulies, "Accurate and comprehensive sequencing of personal genomes," *Genome Res.*, vol. 21, no. 9, pp. 1498–1505, 2011.
[65] E. S. Lander and M. S. Waterman, "Genomic mapping by fingerprinting random clones: A mathematical analysis," *Genomics*, vol. 2, no. 3, pp. 231–239, 1988.
[66] Pacific BioSciences. Accessed: Dec. 1, 2019. [Online]. Available: http://www.pacb.com/products-and-services/pacbio-systems/
[67] D. Branton *et al.*, "The potential and challenges of nanopore sequencing," *Nat. Biotechnol.*, vol. 26, no. 10, pp. 1146–1153, 2008.
[68] 10xGENOMICS. Accessed: Dec. 1, 2019. [Online]. Available: https://www.10xgenomics.com/genome/
[69] M. G. Ross *et al.*, "Characterizing and measuring bias in sequence data," *Genome Biol.*, vol. 14, no. 5, 2013, Art. no. R51.
[70] Y. Turakhia, G. Bejerano, and W. J. Dally, "Darwin: A genomics co-processor provides up to 15,000x acceleration on long read assembly," in *Proc. ACM Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2018, pp 199–213.

S. Karen Khatamifard received the BSc degree in electrical engineering from the Sharif University of Technology, Tehran, Iran, in 2013. He is currently working toward the PhD degree with the Department of Electrical Engineering, University of Minnesota, Minneapolis. His research interests include improving energy efficiency of computing, designing application-specific hardware accelerators, approximate computing, and reliability implications of process technology scaling.

Zamshed Chowdhury (Member, IEEE) received the BSc and MS degrees in applied physics, electronics and communication engineering from the University of Dhaka, Bangladesh. He is currently working toward the PhD degree with the Department of Electrical and Computer Engineering, University of Minnesota, Twin Cities, USA. He is currently a faculty member (on leave) with Jahangirnagar University, Bangladesh. His research interests include emerging non-volatile memory technologies, application specific hardware design, and secure hardware architecture.

Nakul Pande (Student, Member) received the BTech degree in electronics and communication engineering from the Institute of Engineering and Technology, Lucknow, India, in 2010 and the MTech degree in electrical engineering from the Indian Institute of Technology Bombay, Mumbai, India, in 2013. He is currently working toward the PhD degree in electrical engineering with the University of Minnesota, Minneapolis, MN, USA. From 2013 to 2015, he was with Qualcomm Inc., Bangalore, India, where he worked in the areas of design for testability and memory design. His current research interests include different aspects of integrated circuit reliability, ranging from wearout mechanisms at the device level to interconnect reliability, and radiation induced soft errors.

Meisam Razaviyayn received the MS degree in mathematics and the PhD degree in electrical engineering, with minor in computer science, from the University of Minnesota. He is currently an assistant professor of industrial and systems engineering and computer science with the University of Southern California. He was a postdoctoral research fellow with the Department of Electrical Engineering, Stanford University. His research interests include the design and analysis of large scale optimization algorithms arising in modern data science era. He was the recipient of the IEEE Data Science Workshop Best Paper Award in 2019, the Signal Processing Society Young Author Best Paper Award in 2014, and the finalist for Best Paper Prize for Young Researcher in Continuous Optimization in 2013 and 2016.



Chris Kim (Fellow, IEEE) received the BS and MS degrees from Seoul National University and the PhD degree from Purdue University. In 2004, he joined the University of Minnesota, where he is currently a professor. He has authored or coauthored more than 200 journal and conference papers. He was a technical program committee member of several circuit design and semiconductor device conferences. His group has expertise in digital, mixed-signal, and memory IC design, with emphasis on circuit reliability, hardware security, memory circuits, radiation effects, time-based circuits, beyond-CMOS technologies, and machine learning hardware design. He was the recipient of the University of Minnesota's Taylor Award for Distinguished Research, SRC Technical Excellence Award for his "Silicon Odometer" research, Council of Graduate Students Outstanding Faculty Award, NSF CAREER Award, Mcknight Foundation Land-Grant Professorship, 3M Non-Tenured Faculty Award, DAC/ISSCC Student Design Contest Award (2 times), IBM Faculty Partnership Award (3 times), the IEEE Circuits and Systems Society Outstanding Young Author Award, the ICCAD Ten Year Retrospective Most Influential Paper Award, ISLPED Low Power Design Contest Award (4 times), and ISLPED Best Paper Award (2 times).

Ulya R. Karpuzcu received the MS and PhD degrees in computer engineering from the University of Illinois, Urbana-Champaign. She is currently an associate professor with the Department of Electrical and Computer Engineering, University of Minnesota. Her research interests include energy-efficient computing, application domain specialized architectures, approximate computing, and unconventional computing paradigms.