GenStore: A High-Performance In-Storage Processing System for Genome Sequence Analysis

Nika Mansouri Ghiasi
ETH Zürich
Switzerland

Jisung Park
ETH Zürich
Switzerland

Harun Mustafa
ETH Zürich
Switzerland

Jeremie Kim
ETH Zürich
Switzerland

Ataberk Olgun
ETH Zürich
Switzerland

Arvid Gollwitzer
ETH Zürich
Switzerland

Damla Senol Cali
Bionano Genomics
USA

Can Firtina
ETH Zürich
Switzerland

Haiyu Mao
ETH Zürich
Switzerland

Nour Almadhoun Alserr
ETH Zürich
Switzerland

Rachata Ausavarungnirun
KMUTNB
Thailand

Nandita Vijaykumar
University of Toronto
Canada

Mohammed Alser
ETH Zürich
Switzerland

Onur Mutlu
ETH Zürich
Switzerland

ABSTRACT

Read mapping is a fundamental step in many genomics applications. It is used to identify potential matches and differences between fragments (called *reads*) of a sequenced genome and an already known genome (called a *reference genome*). Read mapping is costly because it needs to perform *approximate string matching (ASM)* on large amounts of data. To address the computational challenges in genome analysis, many prior works propose various approaches such as accurate *filters* that select the reads within a dataset of genomic reads (called a *read set*) that *must* undergo expensive computation, efficient heuristics, and hardware acceleration. While effective at reducing the amount of expensive computation, all such approaches still require the costly movement of a large amount of data from storage to the rest of the system, which can significantly lower the end-to-end performance of read mapping in conventional and emerging genomics systems.

We propose *GenStore*, the first in-storage processing system designed for genome sequence analysis that greatly reduces both data movement and computational overheads of genome sequence analysis by exploiting low-cost and accurate *in-storage filters*. GenStore leverages hardware/software co-design to address the challenges of in-storage processing, supporting reads with 1) different properties such as read lengths and error rates, which highly depend on the sequencing technology, and 2) different degrees of genetic variation compared to the reference genome, which highly depends on the genomes that are being compared. Through rigorous analysis of read mapping processes of reads with different properties and degrees of genetic variation, we meticulously design low-cost hardware accelerators and data/computation flows inside a NAND flash-based solid-state drive (SSD). Our evaluation using a wide range of real genomic datasets shows that GenStore, when implemented in three modern NAND flash-based SSDs, significantly improves the read mapping performance of state-of-the-art software (hardware) baselines by 2.07-6.05× (1.52-3.32×) for read sets with high similarity to the reference genome and 1.45-33.63× (2.70-19.2×) for read sets with low similarity to the reference genome.

CCS CONCEPTS

• **Computer systems organization → Special purpose systems**;
• **Hardware → External storage**.

KEYWORDS

Read Mapping, Filtering, Genomics, Storage, Near-Data Processing

ACM Reference Format:
Nika Mansouri Ghiasi, Jisung Park, Harun Mustafa, Jeremie Kim, Ataberk Olgun, Arvid Gollwitzer, Damla Senol Cali, Can Firtina, Haiyu Mao, Nour Almadhoun Alserr, Rachata Ausavarungnirun, Nandita Vijaykumar, Mohammed Alser, and Onur Mutlu. 2022. GenStore: A High-Performance In-Storage Processing System for Genome Sequence Analysis. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '22), February 28 – March 4, 2022, Lausanne, Switzerland.* ACM, New York, NY, USA, 20 pages. https://doi.org/10.1145/3503222.3507702

1 INTRODUCTION

Genome sequence analysis, which analyzes the DNA sequences of organisms, is important for many applications in personalized medicine [1–8], outbreak tracing [9–14], and evolutionary studies [15–21]. The information of an organism's DNA is converted to digital data via a process called *sequencing*. A sequencing machine extracts the sequences of DNA molecules from the organism's sample in the form of strings consisting of four *base pairs (bps)*, denoted by A, C, G, and T. No current sequencing technology has the capability to read a human DNA molecule in its entirety. Instead, state-of-the-art sequencing machines generate randomly sampled, inexact sub-strings of the original genome, called *reads*. The information about the corresponding location of each read in the complete genome is lost during sequencing in most technologies. State-of-the-art sequencing machines produce one of two kinds of reads. 1) Short read sequencing technologies, such as Illumina [22, 23], produce reads that are highly accurate (99-99.9%) [24–26], but short (e.g., up to a few hundred DNA *base pairs* [24, 27, 28]). 2) Long read sequencing technologies, such as Pacific Biosciences (PacBio) [29] and Oxford Nanopore Technologies (ONT) [30], produce reads that are less accurate (85-90%) [27, 31–33], but long (e.g., lengths ranging from thousands to millions of base pairs [34]).

Many genomics applications that involve the comparison of the genomic reads to a reference genome require a fundamental initial process, called *read mapping*. Read mapping identifies potential matching locations of reads against a reference genome [35, 36] and is a *very* computationally-costly process [37–46] due to two key challenges. First, it uses *computationally-expensive algorithms* involving approximate string matching (ASM) [40–42, 44, 45, 47–52]. A read is *aligned* to a reference genome if the read is sufficiently similar to one or more subsequences in that reference genome. Reads generated by a sequencing machine might have differences compared to the reference genome due to either errors in the sequencing process or genetic variations [37, 53–55]. ASM is widely used in existing read mappers to accurately account for such potential differences when determining the similarity between each read and the reference genome. Second, read mapping performs *large amounts of expensive ASM computation* because the genomic read datasets contain many reads (e.g., millions of reads), and each read requires ASM computation on multiple subsequences in the reference genome (see Section 2.1 for more details).

Since read mapping is a key performance bottleneck in genome sequence analysis applications, there has been significant effort into improving read mapping performance via both algorithmic and system optimizations. Many prior works propose efficient heuristics for ASM [56–60], hardware accelerators [35, 39, 40, 61–93], and various *filters* that try to efficiently and accurately prune reads that do not require expensive computation [35–38, 40–43, 45, 48, 49, 94–98]. For example, filters can be used to quickly prune reads that have exact matches in the reference genome. Pruned reads do not go through the expensive ASM process, which improves read mapping performance and efficiency [35, 36, 40–43, 45, 48, 49, 94, 96].

While prior works improve read mapping performance, to our knowledge, *none* of them consider the I/O cost that most systems must pay to read the large amount of data from the storage system to main memory and computation units. Read mapping incurs unnecessary data movement from the storage system for large amounts of *low-reuse* data. For example, while existing filters prune many reads to avoid expensive computation, they still need to first read the *entire* read set from the storage system, even though a large fraction of the reads would be filtered out and *not* be reused in the later stages of the read mapping process. The unnecessary data movement from the storage system can bottleneck read mapping performance in both conventional (software-based) and emerging (hardware-accelerated) genomics systems, while having a larger impact on emerging systems that greatly reduce the computation bottlenecks of ASM (e.g., [35, 39, 40, 43, 61–64]).

In-storage filtering can be a fundamental solution for reducing the cost of the unnecessary data movement in read mapping. Our motivational study using an ideal in-storage filter for read mapping (Section 3) demonstrates that in-storage processing can greatly accelerate the end-to-end read mapping process. This is because in-storage filtering not only avoids unnecessary data movement from storage, but also *eliminates* the computational burden of the filtering process from the rest of the system.

Our goal in this work is to improve the performance of genome sequence analysis by effectively reducing unnecessary data movement from the storage system. To this end, we propose *GenStore*, the first in-storage processing system designed for genome sequence analysis. **The key idea** of GenStore is to exploit low-cost *in-storage accelerators* to accurately *filter out* the reads that do not require the expensive ASM computation in read mapping, thereby significantly reducing unnecessary data movement from the storage system to main memory and processors.

We identify two key challenges in designing an efficient in-storage system for read mapping. First, read mapping workloads exhibit fundamentally different behavior due to 1) the varying read properties such as read length and error rates, which highly depend on the sequencing technology, and 2) the genetic variation of reads compared to the reference genome, which highly depends on the genomes that are being compared. Second, existing filtering methods incur a large number of random accesses to large datasets, which is challenging for a modern NAND flash-based solid-state drive (SSD)[1] to cope with due to its poor random-access performance and limited size of internal DRAM.

We address these challenges with hardware/software co-design in three key directions. First, based on our detailed analysis of read mapping, we design two different accelerators that can accelerate a wide range of read mapping applications for reads with different properties (lengths and error rates) and genetic variations. Each accelerator filters a large fraction of genomic read datasets using simple operations. Second, we develop storage technology-aware algorithmic optimizations to replace expensive random accesses with more efficient sequential accesses to storage devices (e.g., NAND flash-based SSDs). Third, we carefully design an efficient technique for data placement inside the storage device that takes full advantage of the high internal SSD bandwidth to concurrently access large amounts of genomic data.

We design GenStore to support two in-storage filtering mechanisms in a single SSD: 1) GenStore-EM and 2) GenStore-NM. **GenStore-EM** filters *e*xactly-*m*atching reads, i.e., reads that exactly match subsequences of a reference genome. Due to the low error rates of short reads, a large fraction of short reads map *exactly* to the reference genome [43, 55, 99]. For example, on average 80% of human short reads map exactly to the human reference genome [43, 55, 99]. However, finding exactly-matching reads in the SSD is challenging, as it incurs a number of random accesses per read to a large index structure that stores unique subsequences of

[1]In this work, we focus on SSDs based on NAND flash memory, the prevalent memory technology in modern storage systems. We expect that GenStore would also provide performance and energy benefits with storage devices that are built using emerging non-volatile memory technologies.

length *k* (called *k-mers*) and their positions in the reference genome. Since each read consists of many k-mers, filtering each read requires several random accesses to the index. To avoid such random accesses, we introduce a new *sorted, read-sized* k-mer index structure, which enables *sequentially* scanning of the read set and the new index, with only one index lookup per read during filtering.

GenStore-NM filters most of the *non-matching* reads, i.e., reads that would not align to any subsequence in the reference genome. In read mapping, a significant fraction of reads might not align to the reference genome due to 1) the high sequencing error rate (in long reads) and/or 2) high genetic variation (in both short and long reads). For example, both short and long read sets sequenced from rapidly-evolving viral samples (such as SARS-CoV-2) can have high genetic variations compared to the reference genome, leading to, on average, 36% (up to 99.9%) of reads not aligning to the reference [100]. To avoid expensive ASM operations for such non-matching reads, state-of-the-art read mappers commonly employ a step called *chaining*, which calculates a similarity score for each read (called *chaining score*) to the reference and filters out reads with a low score. GenStore-NM uses this basic idea of chaining to build an in-storage filter.

Calculating a chaining score for a read inside the SSD is challenging since it requires performing an expensive dynamic programming algorithm on the read's k-mers that exactly match the reference. This is particularly challenging for long reads since they have a large number of k-mers per read. To avoid such expensive computation, we *selectively* perform chaining only on reads with a small number of exactly-matching k-mers and send other reads to the host system for full read mapping (including chaining). Selective chaining is effective because 1) a read with many exactly-matching k-mers most likely aligns to the reference genome and thus does not require in-storage filtering, and 2) selective chaining can filter many non-aligning reads, without requiring costly hardware resources in the SSD.

To evaluate GenStore, we use a combination of synthesized Verilog models of our in-storage accelerators and state-of-the-art simulation tools that are widely used for DRAM and SSD research, Ramulator [101] and MQSim [102]. To assess the performance impact of the storage system, we evaluate three GenStore-enabled systems with different SSD configurations (low-end, medium-end, and high-end). We integrate GenStore into a state-of-the-art software read mapper (Minimap2 [58]) and two state-of-the-art hardware read mappers (GenCache [43] for short reads and Darwin [39] for long reads). Our results show that GenStore-EM and GenStore-NM improve the performance of Minimap2 by 2.07-6.05× and 1.45-33.63×, respectively, with *no* accuracy loss. GenStore-EM improves the performance of GenCache by 1.52-3.32×, and GenStore-NM improves the performance Darwin by 2.70-19.2×, with *no* accuracy loss.

This work makes the following **key contributions**:

- We introduce GenStore, the first in-storage processing system designed for genome sequence analysis. GenStore fundamentally addresses the high I/O cost of reading low-reuse genomic data from storage systems.
- We address the challenges of in-storage filtering for genome sequence analysis by analyzing the read mapping process and performing hardware/software co-design to develop in-storage filtering mechanisms and accelerators for genomic reads with various lengths, error rates, and genetic variations.
- We introduce two in-storage accelerators, 1) GenStore-EM for filtering exactly-matching reads and 2) GenStore-NM for filtering most reads that would not align to any subsequence in the reference genome. GenStore filters out a large fraction of reads with lightweight hardware accelerators and *no* loss of accuracy, thereby improving the end-to-end performance and energy efficiency of genome sequence analysis.

2 BACKGROUND

We provide brief background on read mapping and NAND flash-based SSDs, necessary to understand the rest of the paper.

2.1 Read Mapping

End-to-End Workflow of Genome Sequence analysis. There are three key initial steps in a standard genome sequencing and analysis workflow [35, 36]. The first step is the collection, preparation, and sequencing of a DNA sample in the laboratory. Modern sequencing machines are unable to read an organism's genome as a single complete sequence; instead they generate shorter subsequences sampled randomly from the genome sequence [103, 104]. The second step is *basecalling*, which converts the representation of the subsequences generated by the sequencing machine (e.g., images or electric current, depending on the sequencing technology [30, 53]) into *reads*, which are sequences of *nucleotides* (i.e., A, C, G, and T in the DNA alphabet). In order to reproduce the complete genome sequence from the shorter read sequences, the third step, called *read mapping*, identifies potential matching locations of each read with respect to a known reference genome (e.g., a representative genome sequence for a particular species) [4, 36, 40, 58]. Genomic read sets can be obtained by, for example, 1) sequencing a DNA sample and storing the generated read set into the SSD of a sequencing machine [105, 106] or 2) downloading read sets from publicly available repositories [107] and storing them into an SSD.

In this work, we focus on optimizing the performance of read mapping because sequencing and basecalling are performed only once per read set, whereas read mapping can be performed *many times* for the same read set. This is common in many genomic applications for two reasons. First, some applications require analyzing the genetic differences between a read set belonging to an individual and *many* reference genomes of other individuals. Examples of such applications include measuring the genetic diversity in a population [108, 109] and determining the donor of a sample by quantifying the reads that have a match in each reference genome [110, 111]. Second, some other applications require repeating the read mapping step many times to improve the outcome of read mapping. Examples of such applications are 1) mapping with new, more updated reference genomes [44], or 2) using different mapping parameter values (such as the maximum number of allowed differences between a read and a subseqeuence in the reference genome so that they are considered similar) [112].

Improving read mapping performance is critical since it is a fundamental step used in almost all genomic analyses that use sequencing data [35, 39, 40, 45]. The contribution of read mapping to the entire analysis pipeline varies depending on the application.

For example, read mapping takes up to 1) 45% of the execution time when discovering sequence variants in cancer genomics studies [113], and 2) 60% of the execution time when profiling the species composition of a multi-species (i.e., *metagenomic*) read set [110].

Read Mapping Process. Since the sequencing process does not provide location information for both short and long reads in most technologies, *read mapping* is a fundamental initial process for many genomics applications. The read mapping process identifies subsequences in the reference genome to which the input reads match. For each *matching location*, i.e., the location of each matching subsequence in the reference genome, a read mapper computes an *alignment score*, indicating the degree of similarity between the read and the region of the reference to which the read aligns. Matching base pairs between the read and the reference increase the alignment score, whereas *edits* (i.e., base pair mismatches, insertions, or deletions relative to the reference) decrease this score.

Since each read is much shorter than the reference genome (e.g., the human reference genome contains ~3.2 billion base pairs), a read mapper typically uses an index of the reference genome to reduce the search space for each read. The index is a dictionary, i.e., a key-value store, where the keys are unique k-length subsequences (called *k-mers*) extracted from the reference genome, and the values are the exactly-matching locations of these k-mers in the reference genome [37, 114]. The value of k is fixed during indexing and used for all subsequent steps.[2] To greatly reduce the storage overhead of the index and speed up queries against it, without significantly changing the final outcome of read mapping, some read mappers index only a subset of reference genome k-mers called *minimizers* [117–119]. A minimizer is a representative k-mer of a set of k-mers according to a scoring mechanism. For example, some read mappers [58, 120] calculate hash values for all k-mers in a window of w consecutive k-mers from an input sequence, and mark the k-mer with the smallest hash value as the minimizer k-mer.

Read mapping is a three-step process. In the first step (*seeding*), the read mapper queries the index structure to determine potential locations in the reference genome where the read could map. To do so, the read mapper looks up every minimizer k-mer fetched from a read in the reference index. If the minimizer k-mer hits in the reference index, the read mapper marks the locations of such a k-mer in the reference genome as the read's *potential matching locations*, also called *seeds*. In the second step (*seed filtering* and/or *chaining*), the read mapper prunes those potential matching locations in the reference to which the read would not align. If all of the potential matching locations of the read get filtered, the read mapper discards the read from further analysis. The read mapper uses a dynamic programming (DP) algorithm to 1) merge overlapping seeds into longer regions, called *chains* [58], and 2) calculate their corresponding *chaining scores,* which refers to the approximation of the entire read's alignment score in these regions. If the read mapper finds one or more chains with a sufficiently high chaining score (indicating a high degree of similarity to the reference genome), then the read mapper performs the third step. If the read has no chain with a sufficiently high score, the read mapper prunes the read and skips the third step. In the third step (*sequence alignment*), the read mapper determines the exact differences between a read and the reference genome at the potential matching locations. Sequence alignment is done with a computationally-expensive DP algorithm [35, 36, 40–42, 45, 48–51] to perform approximate string matching (ASM). Finally, the read mapper returns the locations in the reference genome with the best alignment scores for each read.

Pre-Alignment Read Filtering. To mitigate the high performance overhead of alignment, read filtering approaches are widely used. Read filters can be incorporated at any stage of the process before alignment. There are two main filter types. The first filter type [37, 38, 42, 45, 48, 49, 96] aims to efficiently filter potential matching locations in the reference genome that lead to a large number of edits (larger than a user-defined threshold) between the read and the reference genome at those locations. Doing so avoids a costly alignment step for potential locations at which the read would not match the subsequences of the reference genome. The second filter type [43] aims to detect if a read matches a subsequence of the reference genome with *no* edits (i.e., exact-match) or *very few* (e.g., 1-5) edits. Reads that satisfy this requirement are guaranteed to align to at least one location in the reference genome without requiring the costly read alignment process. This filter type is particularly effective for read sets with a large number of exactly-matching reads (e.g., 80% in human short read sets [43, 55, 99]). While both filter types reduce computation overhead, they still require a large number of random memory accesses for each read, similar to the baseline read mapper. In a typical read set of several gigabytes, read filters incur several random accesses per read 1) to the reference index for seeding, and potentially, 2) to the reference genome to compare the read with the subsequence of reference genome at each potential matching location.

2.2 SSD Organization

Figure 1 depicts the internal organization of a modern NAND flash-based solid-state drive (SSD) that consists of three main components: ❶ NAND flash packages, ❷ an SSD controller, and ❸ DRAM.

NAND Flash Memory. A NAND package comprises multiple *dies* (also called *chips*) that share the package's I/O pins. One or more packages share command/data busses (called *channels*) to connect to the SSD controller. Dies sharing the same channel can operate independently of each other, but only one die can communicate with the SSD controller (e.g., for data transfer) at a time via the shared channel. A die has multiple (e.g., 2 or 4) *planes*. Each plane contains thousands of *blocks*. A block includes hundreds to thousands of *pages*, each of which is 4–16 KiB in size. NAND flash memory performs read/write operations at page granularity but erase operations at block granularity. Planes in the same die share the peripheral circuitry used to access pages; as such, they can concurrently operate only when accessing pages (or blocks) at the same offset, which are called *multi-plane* operations.

SSD Controller. An SSD controller has two main components: 1) multiple cores to run SSD firmware, commonly called the *flash translation layer (FTL)*, and 2) per-channel hardware flash controllers for request handling and error-correcting codes (ECC) for underlying NAND flash chips. The FTL is responsible for communication with the host system, internal I/O scheduling, and various SSD management tasks required for hiding the unique characteristics of NAND flash memory from the host system. For example, a page of NAND flash memory needs to first be erased before it is

[2] k is typically between 11 and 31 [58, 115, 116], depending on the application.

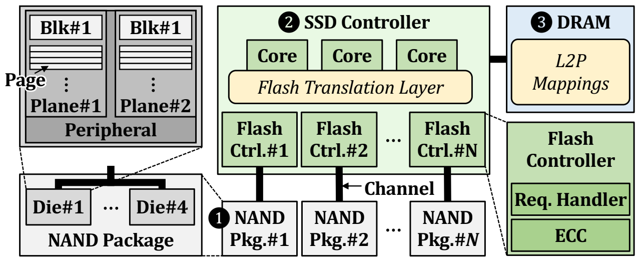


Figure 1: Organizational overview of a modern SSD.

programmed,[3] so the FTL always performs *out-of-place* updates by writing the new data of a *logical* page to a new *physical* page that was erased previously. To this end, the FTL maintains logical-to-physical (L2P) address mappings for reads and performs garbage collection to reclaim new physical pages for writes.

Internal DRAM. A modern SSD employs large low-power DRAM (e.g., 4GB LPDDR4 DRAM for a 4TB SSD [121]) to store metadata for SSD management tasks. Most of the DRAM capacity is used to store the L2P mappings for address translation. It is common practice to maintain the L2P mappings at 4KiB granularity to provide high random access I/O performance [122, 123], so in a 32-bit architecture, the memory overhead for the L2P mappings is approximately 0.1% of the SSD capacity (4 bytes per 4KiB data).

SSD I/O Bandwidth. To mitigate the large performance gap between main memory and the storage system, SSD manufacturers increase the external bandwidth of SSDs by employing advanced I/O interfaces between the host system and SSDs. For example, while older SATA3 SSDs provide around 500MB/s sequential-read bandwidth [121, 124], state-of-the-art PCIe-Gen4 SSDs can provide significantly higher sequential-read bandwidth, up to 8 GB/s (e.g., 7 GB/s in Samsung PM1735 [125]).

A modern SSD's *internal* bandwidth (i.e., I/O bandwidth between NAND flash chips and SSD controller) is usually higher than its external bandwidth (i.e., I/O bandwidth between the host and the SSD). For example, a recent enterprise SSD controller [126] supports 6,550MB/s external bandwidth and 19.2GB/s internal bandwidth (16 channels, each with a bandwidth of 1.2 GB/s). Over-provisioning the internal bandwidth is reasonable since 1) a modern SSD needs to perform various internal management tasks (e.g., garbage collection [127–131] and wear-leveling [129, 132]), and 2) a higher channel count reduces contention between requests by interleaving data between the channels.

3 MOTIVATIONAL STUDIES

We perform experimental studies to understand the potential of efficient in-storage accelerators for improving the performance of genome sequence analysis applications.

3.1 Methodology

Read Mappers. We evaluate five read mapping systems, each of which adopts different optimization techniques to accelerate read mapping: 1) Base uses Minimap2 [58], a state-of-the-art software tool for read mapping. 2) SW-filter extends Minimap2 to *filter out exactly-matching reads* (i.e., reads that exactly match subsequences in one or more locations in the reference genome) using simple single-instruction-multiple-data (SIMD) operations, without requiring costly ASM operations. 3) Ideal-ISF uses an ideal In-Storage Filter (*ISF*) that can *concurrently* filter out exact-matching reads inside the SSD while the host CPU performs read mapping for non-filtered reads. 4) ACC uses a state-of-the-art hardware accelerator for short read mapping, GenCache [43]. 5) Ideal-ISF+ACC uses an ideal in-storage filter (ISF) that can *concurrently* filter out exactly-matching reads inside the SSD while a hardware accelerator (ACC) performs read mapping for non-filtered reads.

System Configuration. To assess the impact of the storage subsystem on end-to-end application performance, we evaluate each of the five systems with four different configurations: 1) a low-end SSD (SSD-L) [124] with a SATA3 interface [133], 2) a mid-end SSD (SSD-M) [134] using a PCIe Gen3 M.2 interface [135], 3) a high-end SSD (SSD-H) [125] with a PCIe Gen4 interface [136], and 4) a system where *all* of the processed data is pre-loaded to DRAM with no performance cost for pre-loading (DRAM), as the *idealized* case where storage I/O overheads are completely eliminated (we do not evaluate DRAM for Ideal-ISF and Ideal-ISF+ACC since using in-storage processing is contradictory to pre-loading all the data to main memory). We assume 8 channels for SSD-L and 16 channels for SSD-M and SSD-H, where the maximum bandwidth per channel is 1.2 GB/s. The maximum internal bandwidth is calculated by 1.2 GB/s × channel count. The external bandwidth of SSD-L, SSD-M, and SSD-H for sequential reads is 500 MB/s, 3.5 GB/s, and 7 GB/s, respectively. Hence, the internal bandwidth of SSD-L, SSD-M, and SSD-H for sequential reads is 19.2×, 5.48×, and 2.74× that of its external bandwidth, respectively.

We evaluate Base and SW-filter by running Minimap2 on a high-end server (AMD EPYC 7742 CPU [137] with 1TB DDR4 DRAM). We simulate the performance of the other three systems using our simulation environment that faithfully models system components including DRAM and storage devices (see Section 5). We map all reads of a short read dataset against the human reference genome, where 80% of the reads have one or more exactly-matching subsequences in the reference genome [55, 99].

Key Features of an Ideal In-Storage Filter. We assume two key features for Ideal-ISF and Ideal-ISF+ACC. First, I/O overheads due to limited external SSD bandwidth are *completely* eliminated for filtered reads. Second, the system provides high in-storage filtering performance such that the filtering process can concurrently run in the SSD, and the latency of this filtering process is *fully hidden* by the read mapping of unfiltered reads in the host CPU (Ideal-ISF) or hardware accelerator (Ideal-ISF+ACC). We assume that the accelerator or the CPU streams through the input reads in batches and analyzes a batch *concurrently* with reading the next batch. Thus, the execution time of Ideal-ISF (+ACC) can be modeled as follows:

$$T_{\text{Ideal-ISF}} = T_{\text{I/O-Ref}} + \max\left\{T_{\text{I/O-Unfiltered}}, T_{\text{RM-Unfiltered}}\right\}, \quad (1)$$

where $T_{\text{I/O-Ref}}$, $T_{\text{I/O-Unfiltered}}$, and $T_{\text{RM-Unfiltered}}$ are the latency of reading the reference genome from the SSD into main memory, the latency of reading the unfiltered genomic reads from the SSD, and the latency of read mapping of the unfiltered reads, respectively. For a given input size, $T_{\text{RM-Unfiltered}}$ varies depending on the computation unit used for read mapping (i.e., the host CPU or accelerator), while the I/O-latency values only depend on the SSD configuration.

[3] This is called the *erase-before-write* property.

3.2 Results & Analysis

Figure 2 shows the execution time of read mapping in the five evaluated systems, each with four different storage subsystem configurations. We make four key observations.

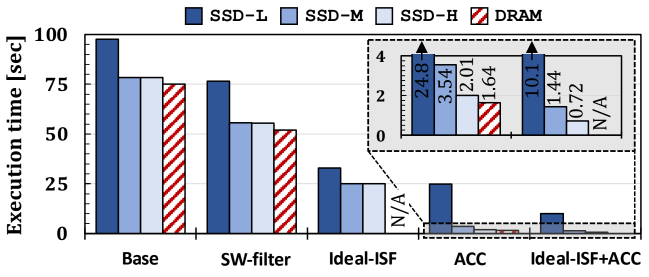


Figure 2: Execution time of read mapping with four different storage configurations.

Observation 1. The ideal in-storage filter provides significant performance improvements over other systems. Ideal-ISF significantly outperforms Base and SW-filter (by 3.12× and 2.21×, respectively), and Ideal-ISF+ACC provides a large speedup (2.78×) over ACC, when they all use SSD-H. These large improvements are due to two key benefits provided by the ideal in-storage filter: 1) mitigation of data movement from the storage devices and 2) removal of the burden of filtering out 80% of the input read set from the rest of the system, including processors and main memory. To distinguish the effects of these two benefits, we analyze an ideal *outside-storage* filter (Ideal-OSF) that provides only the second benefit; this filter concurrently runs with the read mapper and fully overlaps the filtering process with the read mapping process of unfiltered reads. The execution time of Ideal-OSF (+ACC) can be formulated as follows:

$$T_{\text{Ideal-OSF}} = T_{\text{I/O-Ref}} + \max\left\{T_{\text{I/O-All-Reads}}, T_{\text{RM-Unfiltered}}\right\}, \qquad (2)$$

where $T_{\text{I/O-All-Reads}}$ is the latency for reading all genomic reads from the SSD into main memory. Using SSD-H, Ideal-OSF leads to an execution time of 1.15 seconds, which is 60% slower than the Ideal-ISF+ACC. This is because $T_{\text{I/O-All-Reads}}$ is significantly larger than both $T_{\text{RM-Unfiltered}}$ and $T_{\text{I/O-Unfiltered}}$ (in Equation (1)).

The remaining observations dive deeper into the effects of the I/O bottleneck on each read mapping system.

Observation 2. In Base and SW-filter, using high-end SSDs significantly improves read mapping performance over low-end SSDs, effectively reducing the storage performance bottleneck that exists in low-end SSDs. For example, using SSD-H instead of SSD-L reduces the execution time of Base and SW-filter by 24% and 38%, respectively, showing comparable performance to DRAM, where all the data is pre-loaded to main memory (i.e., *no* I/O accesses). This is because, by using SSD-M and SSD-H, the performance bottleneck of the application shifts to parts of the system other than I/O (e.g., CPU or main memory). This observation shows that I/O has a significant impact on application performance but this impact can be alleviated at the cost of expensive storage devices and interfaces. Note that, while SSD-M and SSD-H provide an order-of-magnitude higher bandwidth for sequential reads compared to SSD-L, it is challenging to scale a storage system's capacity using the high-end SSDs due to their significantly-higher prices and the relatively smaller number of the PCIe slots in a server.[4]

Observation 3. Even though SW-filter outperforms Base, its filtering process is slow. Potentially, SW-filter could provide significant performance benefits over Base due to two reasons; 1) as explained, 80% of reads in the dataset exactly match the reference genome, so only 20% of the reads need to undergo the costly ASM computaion; 2) exact-match filtering requires only simple computation, i.e., SIMD XOR operations used by SW-filter. However, even with DRAM, SW-filter's speedup over Base is only 41%. The limited speedup is mainly due to large number of random memory accesses concurrently issued from all threads to the reference index (explained in Section 2.1). This observation highlights the potential of in-storage filtering. Even though both SW-filter and Ideal-ISF filter out the same fraction of reads, the filtering process outside the SSD must compete with the read mapping process for the resources in the system (e.g., the limited main memory bandwidth). In contrast, filtering of reads inside the SSD (where the reads originally reside) can remove the burden of filtering from the rest of the system.

Observation 4. With a hardware accelerator (ACC), using the state-of-the-art SSD (SSD-H) does *not* fully alleviate the storage bottleneck, showing 23% longer execution time compared to when all the data is pre-loaded to main memory (DRAM). While using SSD-M and SSD-H in Base and SW-filter shifts the bottleneck away from I/O, ACC turns I/O into a bottleneck again. This is because ACC greatly reduces the computational bottleneck, which increases the relative effect of the storage subsystem on the end-to-end execution time. The ACC and Ideal-ISF+ACC results clearly show that data movement between the storage devices and the hardware accelerator, which has not been properly considered in prior read mapping accelerators [39, 40, 43, 61, 62, 65, 70–77], can significantly bottleneck the potential benefits of the accelerator.

Comparison to Other Near-Data Processing Systems. Even though read mapping applications could also benefit from other near-data processing (NDP) approaches such as processing-in-main memory (PIM) [45, 65, 140, 141] or processing-in-caches [43], in-storage processing can *fundamentally* address the data movement problem by filtering large, low-reuse data *where the data initially resides.* As an extreme example, even if an *ideal* accelerator achieved a *zero* execution time for read mapping by addressing all of the computation and main memory overheads, there would still exist the need to bring the data from storage to the accelerator. In our motivational study, even SSD-H takes at least 1.55 seconds to read the entire dataset, which is 2.15× slower than the execution time that Ideal-ISF+ACC provides (0.72 seconds). Thus, even though solutions such as processing-in-memory can improve read mapping execution times, they still need to pay the cost of data movement from the storage system to the main memory [39, 40, 45, 61, 65–69, 94]. Therefore, an in-storage filter can be further integrated with any read mapping accelerator, including PIM accelerators, to alleviate their data movement overhead.

[4]The cost of the total storage system depends on both the price of each SSD and the available interconnection slots in the systems. High-bandwidth interconnects such as PCIe take up very large space in the system. As a result, there are fewer PCIe slots than SATA slots in a system. For example, building a 16-TB storage system with a single PCIe SSD (Micron 9300 PRO [138]) costs more than 3,000 USD, while the cost is less than 1,600 USD if we use four 4-TB SATA SSDs (WD BLUE [139]).

3.3 Our Goal

Based on our observations, we conclude that an efficient in-storage filter can be a key enabler for read mappers to achieve high performance in both conventional software-based (e.g., Base and SW-filter) and new hardware-accelerated (e.g., ACC) genomics systems. In particular, in-storage filtering enables the system to take full advantage of the high computation capability of hardware accelerators by fundamentally addressing the data movement bottleneck. **Our goal** is to design an in-storage filter for genome sequence analysis in a cost-effective manner.

We have **three key objectives** in designing our new system. First, the system should provide high in-storage filtering performance to overlap the filtering with the read mapping of unfiltered data (as Ideal-ISF does in our motivation study). Second, it should support reads with 1) different properties (e.g., lengths and error rates) and 2) different degrees of genetic variation in the compared genomes. Third, it should *not* require significant additional hardware overhead, e.g., complicated logic circuits or large SRAM/-DRAM memory.

4 GENSTORE

We propose *GenStore*, the first in-storage processing system tailored for genome sequence analysis. GenStore greatly reduces both data movement and computational overheads of genome sequence analysis by exploiting low-cost and accurate in-storage filters. GenStore supports reads with different properties (lengths and error rates) and different degrees of genetic variation in the compared genomes. We primarily design GenStore as an in-storage accelerator, which is an *extension* of the existing SSD controller and flash translation layer (FTL). GenStore is designed to be integrated into the system such that, when the accelerator is not in use, the entire storage device is available to all other applications, just like in a general-purpose system today.

4.1 Overview

The key idea of GenStore is to exploit low-cost *in-storage* accelerators to accurately *filter out* the reads that do not require the expensive alignment step in read mapping and thus significantly reduce unnecessary data movement from the storage system to main memory and processors. Figure 3 shows the overall architecture of GenStore and how it interacts with the host system. GenStore employs two types of hardware accelerators: ❶ a single *SSD-level* accelerator and ❷ *channel-level* accelerators, each of which is dedicated to a channel. The GenStore-FTL (❸) communicates with the host system and manages the metadata and data flow over the SSD hardware components (i.e., NAND flash chips, internal DRAM, and in-storage accelerators).

Once the host system indicates that the SSD should start analysis as required by a read mapping application (① in Figure 3), GenStore prepares for operation as an accelerator (②). It flushes the conventional FTL metadata necessary to operate as a regular SSD (e.g., L2P mappings [128]), while loading the GenStore metadata necessary for each use case (Section 4.4 provides more details on GenStore FTL). After finishing the preparation, GenStore starts the filtering process. It keeps concurrently reading the data to process from *all* NAND flash chips (③) via multi-plane operations (i.e., it exploits

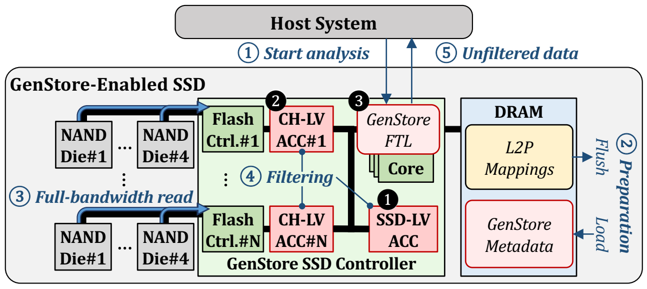


Figure 3: Overview of GenStore.

the SSD's full internal bandwidth, which is much higher than the I/O bandwidth between the SSD and host system [142, 143]), while filtering out reads (④) that do not have to undergo further analysis (e.g., ASM computation). Doing so is possible due to multiple channel-level accelerators that provide computational throughput matching the SSD's internal bandwidth even for the most complicated computation required for filtering. The host system performs further computation as soon as GenStore sends unfiltered reads (⑤), which removes GenStore's filtering process almost completely from the critical path of the application.

As explained in Section 2.2, most of the internal DRAM is occupied by the regular L2P mapping. Therefore, flushing the regular L2P mapping data into NAND flash memory enables GenStore to exploit most of the GB-scale DRAM (e.g., 4GB DRAM in a 4TB SSD [121]) for its operations, which significantly reduces the overhead of additional internal DRAM that might otherwise be required to store the GenStore metadata necessary for the filtering process. We carefully design the GenStore filtering algorithms to only *sequentially* access the underlying NAND flash chips, so GenStore requires only a small amount of metadata to access the stored data. Therefore, GenStore can use most of the internal DRAM space for such metadata. We envision that all GenStore metadata are built *offline* by the host or some other system (e.g., by the sequencing machine when the read set or reference genome are initially stored to the SSD). Constructing GenStore metadata is a *one-time* preprocessing step that can be performed independently of the read mapping process, while the result of the preprocessing step can be used multiple times for different genomics applications.

There exist two main challenges in designing GenStore as an efficient in-storage filter for read mapping. First, the behavior and data-access patterns in read mapping significantly vary depending on the read properties (length and error rate) and genetic variation between the compared genomes. Second, hardware resources (e.g., CPU and DRAM) are quite limited even in modern high-end SSDs. We address these challenges via thorough hardware/software co-design tailored for filtering 1) exactly-matching reads, i.e., reads that exactly match subsequences of the reference genome (Section 4.2), and 2) most of the non-matching reads, i.e., reads that would not align to any subsequence of the reference genome (Section 4.3).

4.2 GenStore-EM for Exactly-Matching Reads

4.2.1 Approach Overview. GenStore-EM accelerates read mapping by using an efficient in-storage filter for reads that have at least one exact match in the reference genome. Due to the low error rates of short reads, combined with low genetic variation between the compared genomes, a large fraction of short reads map *exactly* to the reference genome [43, 55, 99]. For example, on average 80% of human short reads map *exactly* to the human reference genome [43, 55, 99]. Since exact-match detection is computationally cheaper than ASM, concurrently filtering exact-matching reads inside the SSD can significantly improve the runtime of read mapping (as we demonstrate in Section 3). Note that, GenStore-EM is not applicable to long reads due to their greater length. For example, for a 10K base pair-long human read, even with zero sequencing error rate, the probability of the read exactly matching a subsequence in the reference genome is very low (e.g., $< 3.6 \times 10^{-6}$) due to natural genetic variation.[5]

Key Challenges. The key challenge in designing GenStore-EM is the large number of random accesses to large data structures inside the SSD. As explained in Section 2, identifying exact matches for read mapping requires a number of random accesses *for each k-mer in a read* to two large data structures: 1) a large k-mer index, to find potential matching locations of each k-mer in the reference genome, and 2) the reference genome, to find candidate matching sequences at the candidate matching locations in the reference genome. Handling random accesses to large data structures is challenging for NAND flash-based SSDs for two reasons. First, NAND flash memory exhibits poor performance for random access reads. Second, we cannot use in-SSD DRAM to store the large data structures that are randomly accessed, since even in high-end SSDs, the size of internal DRAM is relatively small (e.g., 4 GB [121]) compared to the size of the data structures that GenStore-EM needs to handle (e.g., 7 GB for the human reference genome and its index [58]).

Key Idea. The key idea in GenStore-EM is to *sequentialize* most data accesses via a carefully designed metadata structure and data layout. To do so, we design a new *sorted, read-sized* k-mer index structure. This index enables *sequential* scanning of the read set and the new index, with only one index lookup per read while performing filtering.

Figure 4 shows the key idea of GenStore-EM with a simplified example in which each short read consists of three base pairs (bps).[6] Suppose that we have two data structures: 1) a sorted read table (*SRTable*), each entry of which stores a read and its unique ID, and 2) a sorted k-mer index (*SKIndex*), which contains *all unique read-sized* k-mers of the reference genome, along with each k-mer's corresponding locations in the reference genome. Each data structure is *sorted* by read/k-mer in alphabetical order. With these two data structures, it is possible to identify each read's exactly-matching locations in the reference genome by *streaming* both reads and k-mers through a simple comparator. We use two pointers, *r* and *k*, which point to the current entries we are examining in SRTable and SKIndex, respectively. We *sequentially increment* the two pointers in three different ways based on the comparison result of the current read and k-mer. First, when the current read and k-mer are identical (❶ in Figure 4), we record the read as an exactly-matching read and increment *r* and *k*. Second, if the read is alphabetically larger than the k-mer (❷), we conclude that the k-mer does not match any read in SRTable and increment *k* (so that we can examine if the

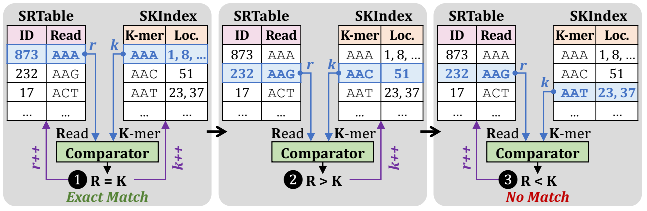


Figure 4: Overview of the key idea of GenStore-EM.

next k-mer matches the read). Third, if the k-mer is alphabetically larger than the read (❸), we conclude that the read does not match any k-mer in SKIndex. We record the read as *not* an exact match and increment *r* (so that we can examine the next read).

GenStore-EM's two data structures and filtering algorithm enable an exact-match filter highly suitable for in-storage processing. First, since these two data structures are only sequentially accessed, the filtering process can be done in a *streaming* manner, leveraging the high sequential read bandwidth of NAND flash memory. Second, we can easily perform exact-match detection of a read and a read-sized k-mer with simple comparator logic and fully pipeline the filtering process with sequential access to the data structures.

A *read-sized* k-mer index increases the total amount of accessed data for read mapping. The reason is that a read-length value of k (e.g., k=150) significantly increases the number of unique k-mers, compared to the k values commonly used in conventional read mappers (e.g., k=15). For example, the size of an index structure for all unique k-mers in the human reference genome is 21 GB when k=15, while the size increases to 126 GB when k=150. However, our proposal (i.e., a large yet sequentially-accessed SKIndex) is feasible and desirable for in-storage processing due to the large capacity and high internal bandwidth of modern NAND flash-based SSDs.

4.2.2 *Design of GenStore-EM.*

Figure 5 illustrates the overall operational flow of GenStore-EM, which consists of two steps: **Step 1.** data fetching and **Step 2.** exact-match filtering. GenStore-EM uses the two data structures explained in Section 4.2.1: 1) a sorted read table (SRTable) for storing the read set and 2) a sorted k-mer index (SKIndex) for storing read-sized k-mers from the reference genome. Step 1 reads the two data structures from NAND flash chips to the SSD's internal DRAM in a batched manner (❶ in Figure 5). Step 2 performs exact-match filtering within each read batch, using simple comparator logic in the SSD-level accelerator (❷). Steps 1 and 2 are performed in a pipelined manner. During filtering, GenStore-EM sends the unfiltered reads to the host system for full read mapping. This enables the concurrent filtering of reads in the SSD and read mapping of the unfiltered reads in the host system.

Data Structures. We carefully design SRTable and SKIndex to minimize performance and storage overheads of GenStore-EM, by extending the two data structures described in Figure 4 in two aspects. First, both SRTable and SKIndex contain a *strong* hash value (e.g., SHA-1 [145] or MD5 [146]) of each read and read-sized k-mer, respectively, which is used as both the sorting criterion of the data structures and a *fingerprint* of each read and k-mer that is used by the comparator logic. Second, unlike the data structures described in Figure 4, SKIndex no longer contains the k-mers of the reference genome but only the fingerprints of the k-mers. Using strong hash values enables GenStore-EM to determine exact matches between

[5]A typical human genome contains genetic variations at ~4.1 to 5 million [55] out of a total of ~3.2 billion base pairs [144]. Thus, each 10K-bps read contains, on average, ~12.5 to 15.3 base pairs that are different from the reference genome.

[6]In a realistic scenario, the read length is much larger (e.g., 150 bps).

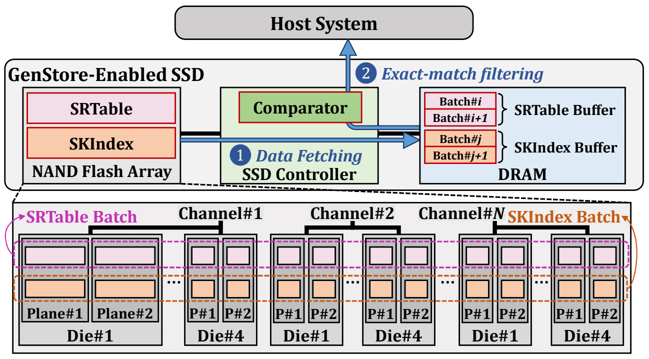


Figure 5: Overview of GenStore-EM.

a read and a k-mer by comparing only their fingerprints, which provides two benefits. First, it reduces the storage overhead of SKIndex by obviating the need to store the raw read-sized k-mers.[7] For the human reference genome, the size of the optimized SKIndex (which stores fingerprints instead of read-sized k-mers) is 32 GB when the read size is 150 bps, which is 3.9× smaller than the size of the unoptimized SKIndex. Second, using fingerprints significantly reduces the performance overhead of exact-match detection by avoiding comparisons of reads and k-mers that are hundreds of bytes in size.

Note that such exact-match detection does *not* affect the accuracy of read mapping due to the extremely low collision rate of strong hash functions. Even in an extremely rare case of a hash collision, the impact of the collision on GenStore's accuracy will be negligible [147], since the DNA information loss due to the falsely filtered read will highly likely be compensated by other reads generated from the neighboring locations in the DNA, which almost *fully* overlap with the falsely filtered read but have totally different hash values. This is because it is common practice to sequence each DNA fragment several times (i.e., with high coverage) to improve the accuracy of downstream genetic analyses [26, 53, 54, 148].

We envision that all GenStore data structures are built *offline* by the host or some other system (e.g., by the sequencing machine when the read set or reference genome are initially written to the SSD). This preprocessing overhead can be hidden by two essential initial steps of the genome sequence analysis pipeline: 1) sequencing and 2) basecalling. For example, in the current highest-throughput Illumina NovaSeq 6000 sequencer [149], sequencing and basecalling work in a pipelined manner and generate genomic read data at a limited throughput of 18.9 MB/s [149]. We analyze the throughput of GenStore's preprocessing step (i.e., generating hash values and sorting reads) and observe that even a personal laptop [150] can provide 174 MB/s of preprocessing throughput. Therefore, GenStore's preprocessing can be done in a pipelined manner with sequencing/basecalling, without decreasing the overall throughput of these steps. This low preprocessing overhead can be further amortized since the preprocessed data can be reused multiple times in different read mapping experiments.

[7] We design SRTable to store the raw reads so that we can transfer *unfiltered* reads to the host for full read mapping after we detect them as non-exactly-matching reads.

Step 1. Data Fetching. GenStore-EM reads SRTable and SKIndex in batches, while exploiting the *full* internal bandwidth of the SSD. In Figure 5, we refer to each batch of SRTable as an *SRTable Batch* and each batch of SKIndex as an *SKIndex batch*. The batch size is equal to the size of data that can be read in parallel by a multi-plane read operation for each chip (i.e., *Number of Planes in the SSD* × *Page Size*), which enables 100% utilization of the NAND flash chips while reading a batch. As shown in Figure 5 (bottom), GenStore-EM stores the SRTable and SKIndex to NAND flash chips in an *interleaved* manner so that each of the data structures can be *sequentially*, *evenly* distributed across all the NAND flash chips.

GenStore-EM exploits the SSD's full internal bandwidth using double buffering. As shown in Figure 5, GenStore-EM employs two sets of batch buffers in the internal DRAM: *SRTable Buffer* (for SRTable) and *SKIndex Buffer* (for SKIndex), each of which can store two batches of the respective data structure. After Step 1 finishes fetching *Batch#i*, it proceeds to fetching *Batch#i* + 1, while Step 2 starts working on *Batch#i*. If the two steps work with the same throughput, we only need to buffer two batches for each data structure. For example, to enable double-buffering in an 8-channel SSD (with four 2-plane dies per channel and 16-KiB pages), the batch buffers require 8MB DRAM space in total.

Step 2. Exact-Match Filtering. Step 2 scans through each batch of SRTable and SKIndex stored in SRTable Buffer and SKIndex Buffer, respectively, comparing the fingerprints (strong hash values) with a simple hardware comparator. When $FP(r_i) > FP(k_j)$ (where $FP(x)$ is the fingerprint of x, and r_i and k_j are the i-th read and j-th k-mer in the current batch, respectively), Step 2 scans SKIndex while increasing j until it finds k_j such that $FP(r_i) \leq FP(k_j)$. If $FP(r_i) < FP(k_j)$, it is guaranteed that no exact match exists for read r_i, so GenStore-EM sends r_i to the host for the read mapping process. When $FP(r_i) = FP(k_j)$, i.e., the two fingerprints are identical, and GenStore-EM marks the read as an exactly matching read.

Due to the simple computation in Step 2, the execution time of GenStore-EM is bottlenecked by Step 1 (Data Fetching). As explained, Step 1 only streams the data structures in batches from the underlying NAND flash chips to the internal DRAM, leveraging the SSD's full internal bandwidth. Therefore, the performance of GenStore-ER can be easily scaled up by increasing the SSD's internal parallelism (e.g., by deploying more channels or using low-latency NAND flash memory [122, 151, 152]).

4.3 GenStore-NM for Non-Matching Reads

4.3.1 Approach Overview. GenStore-NM filters most of the *non-matching* reads, i.e., reads that would not align to any subsequence in the reference genome. This is motivated by the fact that in read mapping, a large fraction of reads might *not* align to the reference genome due to 1) the high sequencing error rate (in long reads) and/or 2) high genetic variation between the compared genomes (in both short and long reads). To illustrate this, we analyze four read mapping use cases, one with read sets with high sequencing error rates, and three with high genetic variation in the compared genomes. Use cases with high genetic variation include 1) samples from rapidly-evolving species (such as SARS-CoV-2 [100]) that have high genetic variation compared to the reference genome, 2) samples with no known reference genomes, and 3) mapping a read set against the human reference genome to filter out *human contamination*, i.e., reads that have been sequenced from human-origin contaminant DNA in a non-human sample.[8] For each use case (except for the *Contamination* use case), we analyze two different combinations of the input read set and the reference. Table 1 summarizes the result of this analysis by showing input read sets, the properties of each read set (e.g., read length and dataset size), reference genomes, and the percentage of reads in each read set that align to subsequences in the reference genome. We observe that a large fraction of reads (31.7%–99.6%) within a read set does *not align to any subsequence* in the reference genome. Quickly filtering this large fraction of non-aligning reads in the SSD can reduce the large data movement from the storage and expensive ASM computation for reads that would not align.

Table 1: Fraction of aligning reads in various read mapping use cases with short and long reads.

Use case	Input read set (Short/Long)	Size [GB]	Reference	Align [%]
Sequencing errors	ERR3988483 (L) [157]	54	hg38 [144]	**47.4**
	HG002_ONT_20200204 (L) [158]	371		**69.3**
Rapidly evolving samples	SRR5413248 (L) [157]	1.69	NZ_NJEX02 [159]	**60.0**
	SRR12423642 (S) [157]	0.466	NC_045512.2 [160]	**23.1**
No reference	SRR6767727 (L) [157]	12.4	NZ_NJEX02 [159]	**0.35**
	SRR9953689 (L) [157]	15.9		**37.0**
Contamination	SRR9953689 (L) [157]	15.9	hg38 [144]	**1.0**

To avoid expensive ASM computation for reads that would not be aligned, state-of-the-art read mappers commonly employ a step called *chaining* (described in Section 2.1), which calculates each read's similarity score (called *chaining score*) to the reference genome and filters out reads with a low score. GenStore-NM uses this basic idea of chaining to build an in-storage filter.

Key Challenges. Calculating a chaining score in the SSD is challenging because finding the best chaining score requires performing an expensive dynamic programming (DP) algorithm on every potential matching location (i.e., seed) within a read (explained in Section 2.1). Normally this operation has $O(N^2)$ time and space complexity, where N is the number of seeds. Even though a chaining score can be approximated accurately in $O(hN)$ time and space by considering only $h < 50$ seeds at a time [58], we observe that some long reads can have up to thousands of seeds due to their length. Therefore, designing a chaining accelerator to perform an expensive DP algorithm on these large reads ($N > 1000$) can incur significant performance or area overheads.

Key Ideas. To avoid such expensive chaining, GenStore-NM selectively performs a fast version of chaining *only* on reads with a small number of seeds and sends other reads to the host system for full read mapping (including complete chaining). This idea is based on our key observation that 1) a read with a large number of seeds most likely aligns to the reference genome and does not require in-storage filtering, and 2) selective chaining can filter many non-aligning long reads, without requiring costly hardware resources in the SSD. Figure 6 shows the alignment probability of a read in a long read dataset (SRR5413248 [161] in Table 1) to subsequences in the reference genome (NZ_NJEX02 [159]), as a function of the number

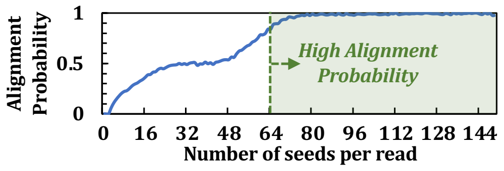


Figure 6: Alignment probability as a function of the number of seeds per read in a long read mapping use case.

of seeds per read (N). The average read length in this dataset is 10K base pairs and the number of seeds goes up to several thousands for some reads. We observe that reads with a sufficiently large number of seeds are very likely to align to subsequences in the reference genome (e.g., at least 85% of reads with $N \geq 64$ seeds align). Such reads can be directly sent to the CPU for full read mapping (bypassing the in-storage chaining-based filter).

We extend this analysis to read datasets from various organisms commonly used in genomics studies: *E. coli* [159], yeast [162], thale cress [163], fruit fly [164], mouse [165], and human [144]. We observe that when a read has $N = 64$, 128, or 256 seeds, it aligns with average probabilities of 88.87%, 91.32%, and 93.84%, respectively. Based on these observations, we design GenStore-NM to *selectively* perform chaining only on reads with fewer than N seeds, while sending reads with at least N seeds to the host system.[9] This selective chaining significantly reduces the chaining execution time and additional hardware area cost, while filtering most reads that would not align to the reference genome.

4.3.2 *Design of GenStore-NM.*

Figure 7 shows the overview of GenStore-NM that filters out most of the non-matching reads in three steps. In Step 1, GenStore-NM reads the input read set from the flash chips, generates minimizer k-mers for each read (as explained in Section 2.1), and looks up each minimizer in a *K-mer Index* (KmerIndex) to find the potential matching locations, i.e., seeds (❶ in Figure 7). In Step 2, GenStore-NM counts the number of seeds in each read to decide if the read needs to go through chaining (❷). To further improve overall performance, Step 2 also filters out reads with *too few* seeds (i.e., $< M$), which would not align to the reference and thus would be filtered anyway by the baseline read mapper [58]. In Step 3, GenStore-NM filters reads based on their chaining scores using a fast and efficient chaining accelerator (❸). If a read has at least one chain with a score above a specified threshold, it is sent to the CPU for mapping. Otherwise, the read is filtered. All three steps run in a pipelined manner.

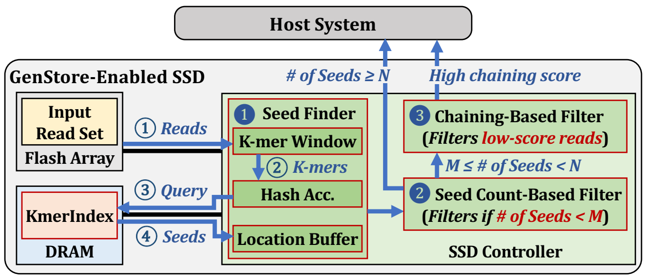


Figure 7: Overview of GenStore-NM.

[8]Contamination of non-human samples with human DNA is commonly observed [153, 154] and corrected for [154–156].

[9]GenStore-NM's design can be tuned based on different values of N.

Data Structures. We carefully design the KmerIndex to reduce the SSD's internal DRAM capacity required for storing it. KmerIndex, similar to the index used by the baseline read mapping tool (i.e., Minimap2 [58]), is a hierarchical hash table. To reduce the memory capacity required for storing the KmerIndex, we make three modifications: 1) not store the reference genome since it is not needed by our approach, 2) not store seeds with *many* matching locations (e.g., more than 495 locations in the baseline read mapper [58])[10] since read mappers usually ignore such seeds in the chaining process [58], and 3) increase the number of hash table buckets so that each bucket holds one minimizer. Increasing the number of buckets increases the false positive rate in the filter, which leads to finding extra seeds and performing extra chaining operations (without loss of accuracy). We make this trade-off to reduce the KmerIndex's size and the required memory space. These optimizations reduce the size of the index for the human reference genome [58] from 5.8 GB to 2.9 GB, which enables GenStore-NM to easily store the KmerIndex in the limited DRAM space inside the SSD.

Step 1: Seed Finding. This step finds the potential matching locations of a read (i.e., seeds) in the KmerIndex. GenStore-NM first reads the input read set from all flash chips (① in Figure 7). GenStore-NM exploits the *full* internal bandwidth of the SSD by reading the read set in parallel via a multi-plane read operation for each chip (same as GenStore-EM, Section 4.2.2). In a shifting buffer (called the *K-mer Window*) in the channel-level accelerator, GenStore-NM stores the w most recently read k-mers of each read. For each sliding window of w k-mers,[11] GenStore-NM finds the minimizer of the window (described in Section 2.1) using a 64-bit Integer Mix hash function (hash64) [166] in the SSD-level accelerator (②). GenStore-NM queries each minimizer in the KmerIndex until it reads N seeds (e.g., N = 64) or it reaches the end of the read (③). Each index query takes up to two memory accesses (to visit the two levels of the hash table). GenStore-NM stores the seeds in the *Location Buffer* in the channel-level accelerator (④), and moves to Step 2.

Step 2: Seed Count-Based Filtering. This step compares the number of seeds in the Location Buffer with a lower bound M and an upper bound N. If a read has fewer than M matching seeds, it is filtered since it will not meet the minimum chaining score requirement of the baseline read mapper [58].[12] If the read has more than N seeds, it means that the read will very likely align to the reference (e.g., reads with at least 64 seeds in Figure 6) and thus require the full read mapping process. GenStore-NM sends such reads to the host system for the full read mapping process. This way, the read mapping process of the unfiltered reads can run concurrently with GenStore's filtering operations. For any other read, GenStore-NM performs chaining in the SSD in Step 3.

Step 3: Chaining-Based Filtering. This step performs chaining (see Section 2.1), a step commonly used in state-of-the-art read mappers [58, 120, 167] to filter out reads with low chaining scores and send reads with high chaining scores to the CPU to undergo the full read mapping process. The key difference in GenStore-NM's chaining process compared to existing read mappers is that GenStore-NM *selectively* performs chaining only on reads with lower seed counts than a threshold N. Such selective chaining is based on our two key observations; 1) a long read with many seeds most likely aligns to the reference genome and thus does not require in-storage filtering, and 2) selective chaining can filter many non-aligning long reads, without requiring costly hardware resources in the SSD (as shown in Section 4.3.1).

We design GenStore-NM's chaining unit based on the chaining algorithm used in Minimap2 [58], a state-of-the-art baseline read mapper. A chain is computed from a sequence of seeds $S_1, \ldots, S_N$[13] of lengths $w_1, \ldots, w_N$, respectively. For a given seed S_i, x_i (y_i) denotes the ending position of the seed's matching location in the reference genome (the read). The chaining score acts as an approximation of an alignment score, increasing linearly with the number of matching base pairs between the read and the reference, and decreasing with the size of gaps between the seeds. The chaining score is defined as follows (based on [58]):

$$f(i) = \max\left\{ \max_{i>j\geq 1} \{f(j) + \alpha(j,i) - \beta(j,i)\}, w_i \right\}, \tag{3}$$

where $f(i)$ is the best chaining score which can be computed with the seeds $S_1, \ldots, S_i$. Given a chain whose last seed is S_j, $\alpha(j,i)$ (also called the *match score*) is the number of new base pairs added to the chain after adding S_i. $\beta(j,i)$ (also called the *gap penalty*) subtracts from the chain score based on the distance between S_i and S_j in the reference genome.[14]

GenStore-NM's Chaining-Based Filter (in Figure 7) consists of 1) a *Chaining Buffer* to store x_i, y_i, w_i, and $f(i)$ values and 2) a *Chaining Processing Element (PE)* based on Equation (3). Figure 8 shows the design of the Chaining PE in the channel-level accelerator. We reduce the latency and size of this unit by approximating multiplication with shift operations. We ensure that our hardware optimizations always over-estimate the chaining score so that we do not filter out any potential read mappings. GenStore-NM does not affect the accuracy of the read mapper because the filter performs the same computations as the baseline chaining filter for reads with seeds $< N$ inside SSD. The chaining PE needs to be executed $N \times M$ times per read where N is the number of seeds per read and M is the DP-iterations of each seed. Due to the limited number of seeds that go through the chaining step, we limit the number of chaining units that are needed to match the full internal bandwidth of SSD.

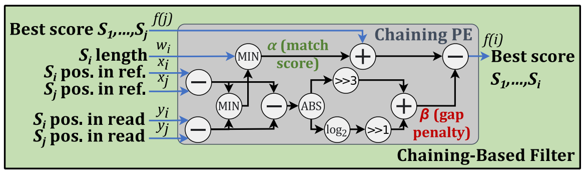


Figure 8: Chaining processing element (PE) in GenStore-NM.

Similar to GenStore-EM, the steps of GenStore-NM are pipelined. The performance of GenStore-NM can be easily scaled up by increasing the SSD's internal parallelism (e.g., by deploying more channels or using low-latency NAND flash memory) [122, 151, 152].

[10] Each matching k-mer can map to one or more locations in the reference genome.

[11] The default value for w in Minimap2 [58] is 10, but GenStore's design can be tuned for different values.

[12] We use M=3, as in [58], but GenStore-NM's design trivially supports different values of M.

[13] Sorted based on their locations in the reference genome.

[14] See Minimap2 [58] for more details regarding $\alpha(j,i)$ and $\beta(j,i)$.

4.4 GenStore Flash Translation Layer (FTL)

GenStore requires simple changes to the existing FTL code.
GenStore FTL Metadata. GenStore metadata includes the mapping information of the data structures necessary for read mapping acceleration, which enables access to the data structures without the L2P mapping table of the regular FTL. In accelerator mode, where GenStore operates as an accelerator, GenStore also keeps in internal DRAM other metadata structures of the regular FTL (e.g., the page status table and block read counts [129, 152]) which need to be updated during the filtering process. We carefully design GenStore to only *sequentially* access the underlying NAND flash chips while operating as an accelerator, so it requires only a small amount of metadata to access the stored data.
Data Placement. GenStore needs to properly place its data structures to enable the full utilization of the internal SSD bandwidth in accelerator mode. When each data structure is initially written to the SSD, GenStore sequentially and evenly distributes it across NAND flash chips. We design GenStore to store every data structure using multiple sets of NAND flash blocks such that each block set consists of NAND flash blocks with the same block offset across the planes in a die. For example, a die's block set k includes all NAND flash blocks whose offset is k in the die. Such data placement enables GenStore not only to always perform *multi-plane* read operations during the filtering process, but also to significantly reduce the size of GenStore metadata. For example, suppose that a GenStore-enabled SSD consists of 128 2-plane NAND flash dies, and the block size is 12 MB (i.e., the size of each block set would be 24 MB). In such a case, GenStore can specify the physical location of a 30-GB data structure by maintaining only the list of 1,250 (30 GB/24 MB) physical block addresses. This way, GenStore significantly reduces the size of the necessary mapping information from 300 MB (with conventional 4-KiB page mapping) to only 5 KB (1,250×4 bytes), and the saved internal DRAM space can be used for loading data structures in the accelerator mode.
SSD Management Tasks. In accelerator mode, GenStore stops working as a regular SSD. It only reads data structures to perform filtering, and does not write any new data. Therefore, GenStore does not require any write-related SSD-management tasks such as garbage collection [127–131] and wear-leveling [129, 132, 168].

The other tasks necessary for ensuring data reliability, such as refreshing data to avoid uncorrectable errors due to data retention and read disturb [129, 168–176], can be done before or after the filtering process, for two reasons. First, GenStore significantly limits the amount of data whose retention age would exceed the manufacturer-specified threshold for reliable operation (e.g., 1 year [177]) during the filtering process, since GenStore's filtering process takes a short time.[15] GenStore simply refreshes such data [129, 170, 173, 174] before starting the filtering process. Second, GenStore-FTL can easily avoid read disturbance errors for data with high read counts [172, 175] since GenStore sequentially reads NAND flash blocks only *once* during filtering. After the filtering process, to prevent read disturb errors in future filtering processes or regular accesses, GenStore refreshes pages that store a data structure if the structure's read count exceeds a certain threshold.

[15] E.g., less than 7 minutes for a 1TiB dataset even with a low-end SSD (SSD-L) [121].

5 EVALUATION METHODOLOGY

Evaluated Systems. We show the benefits of GenStore when it is integrated with the state-of-the-art software and hardware read mappers. To this end, we evaluate the following systems:

- **Base:** Minimap2 [58] is a state-of-the-art software read mapper baseline for both short and long reads. GenCache [43] and Darwin [39] are state-of-the-art hardware read mappers for short and long reads, respectively.
- **GS-Ext:** Base integrated with an implementation of the GenStore filter without in-storage support (Ext stands for *external* to storage). GS-Ext concurrently filters reads while using Base to perform read mapping for unfiltered reads. The goal of evaluating GS-Ext is to decouple the effects of GenStore's two major benefits: 1) alleviating I/O bottlenecks via efficient in-storage processing and 2) reducing the workload of the read mapper by filtering reads with simple operations. GS-Ext obtains the second benefit but not the first. For software read mappers, we evaluate a pure software implementation of GenStore that concurrently runs with Base.[16] For hardware read mappers, we evaluate a hardware implementation of GenStore *outside* the SSD.
- **GS:** Base integrated with the hardware GenStore filtering accelerators, GenStore-EM and GenStore-NM, as described in Sections 4.2 and 4.3. GS concurrently filters reads *inside* the SSD while using Base to perform read mapping for unfiltered reads.

The source code of GenStore and scripts and datasets can be freely downloaded from https://github.com/CMU-SAFARI/GenStore.
Area and Power. We implement GenStore's logic components in Verilog HDL. We synthesize our designs using the Synopsys Design Compiler [178] with a 65nm process technology node to estimate latency, area and power consumption. We use the SSD power values of the Samsung 3D NAND flash-based SSD [121], and DRAM power values based on DDR4 model [179, 180].
Performance Modeling. We evaluate **hardware** configurations using two state-of-the-art simulators to analyze the performance of GenStore. We model DRAM timing with the DDR4 interface [180, 181] in Ramulator [101, 182], a widely-used, cycle-accurate DRAM simulator. We model SSD performance using MQSim [102], a widely-used simulator for modern SSDs. We model the end-to-end throughput of GenStore based on the throughput of each GenStore pipeline stage: accessing NAND flash chips, accessing internal DRAM, accelerator computation, and transferring unfiltered data to the host. We estimate the performance of GenCache and Darwin accelerators based on the data reported in the original works [39, 43].
Real System Results. We use real systems to evaluate all **software** configurations. For a given reference genome, separate reference indexes are generated for each sequencing technology using the software read mapper's default settings for each technology [58]. All other read mapping parameters are kept at their default values. We perform all experiments on an AMD® EPYC® 7742 CPU with 1TB DDR4 DRAM (available in user space). We measure power in these systems using AMD® μProf [183]. We carefully evaluate

[16] We do not evaluate a separate GS-Ext configuration for long read software mapper since Base (Minimap2 [58]) already incorporates the chaining filter used in GenStore-NM. GenStore-NM implements part of this chaining filter at low cost (enabled by the key observations in Section 4.3) to fit within the constraints of in-storage processing.

GenStore's benefits over the baseline in a conservative manner by using optimized configurations for all baselines. Our baselines fetch data from the SSD by sequentially reading the data in batches [58], while providing sufficiently large DRAM capacity to contain all data that gets reused *during* read mapping. We use the number of threads that leads to each software configuration's best performance (i.e., execution time): 128 threads for Base and 16 threads for GS-Ext in our experimental setup.[17]

SSD Configurations. We analyze the benefits of GenStore on three SSD configurations: 1) a low-end SSD (SSD-L) [124] with a SATA3 interface [133], 2) a mid-end SSD (SSD-M) [134] using a PCIe Gen3 M.2 interface [135], and 3) a high-end SSD (SSD-H) [125] with a PCIe Gen4 interface [136].

Datasets. For short read experiments, we use the hg38 human reference genome [144]. In Section 3, we use real short reads (SRR2052419 [158], 19.6 GB). In Section 6, to flexibly and controllably analyze the effect of read sets with different features, we simulate read sets with various sizes (up to 440 GB) and different fractions of exactly-matching reads (75% and 85%) using the Mason 2 genomic read simulator [184]. We generate reads with different exact-match rates by introducing sequence mutations (uniformly randomly drawn from the gold-standard mutation list for the human sample NA12878 [158]) to the reference genome. For the long read experiments, we use the reference genome and read set from Table 1 in Section 4.3.1. To flexibly analyze the effect of data size, we generate larger read sets by concatenating the original read sets several times.

6 EVALUATION

6.1 Area and Power Analysis

We propose GenStore as an in-storage processing system that supports both accurate short reads and error-prone long reads with different levels of genetic variation between compared genomes. Table 2 shows the area and power consumption of each logic unit used in GenStore. In the first column, the table shows the different logic units used in GenStore along with the width of each entry. The second column shows the number of instances of each unit in an 8-channel SSD. We find the number of instances based on the frequency of and the required throughput from each unit such that GenStore can process data concurrently read from all planes in all SSD channels. GenStore's hardware units work at different clock frequencies, with the lowest frequency being 300 MHz (for each channel-level Chaining PE). While the hardware units can be designed to operate at higher frequency, their throughput is sufficient when operating at lower frequencies since the end-to-end execution time of GenStore is bottlenecked by NAND flash reads. The third and fourth columns show the area and power of one instance of each unit.

The total hardware area needed for GenStore is very small (0.20 mm^2 at 65 nm and 0.02 mm^2 at 14 nm, only 0.006% of a 14nm Intel Processor [185]).[18] The area overhead of GenStore hardware

[17]Performance of GS-Ext saturates after 16 threads since it becomes bottlenecked by the external SSD bandwidth.

[18]Since lower technology nodes are not available publicly, we scale the area consumption down to lower process technology nodes using the methodology in [186].

Table 2: Area and power breakdown of GenStore's logic

Logic unit	# of instances	Area [mm^2]	Power [mW]
Comparator (64-bit)	1 per SSD	0.0007	0.14
K-mer Window (10×19-bit)	2 per channel	0.0018	0.27
Hash Accelerator (64-bit)	2 per SSD	0.008	1.8
Location Buffer (64×64-bit)	1 per channel	0.00725	0.37375
Chaining Buffer ($50 \times$ (16-bit + 64-bit))	1 per channel	0.008	0.95
Chaining PE	1 per channel	0.004	0.98
Control	1 per SSD	0.0002	0.11
Total for an 8-channel SSD	-	0.20	26.6

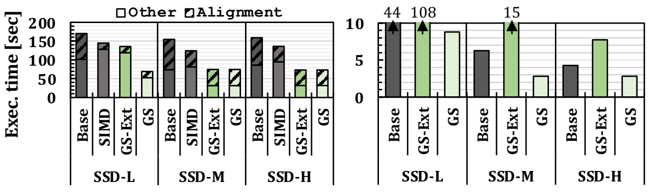
(0.06 mm^2 at 32 nm) is less than 9.5% of the three 28nm ARM Cortex R4 processors [187] in a SATA SSD controller [121].

The area and power of most logic units (except for the comparator, the hash64 accelerator, and the control unit) increase linearly with the channel count. Each instance of the comparator and hash64 accelerator supports multiple channels (up to 12 and 4 channels, respectively). Therefore, these units scale by $\lceil \frac{\#channels}{12} \rceil$ or $\lceil \frac{\#channels}{4} \rceil$, respectively. The area and power of GenStore's control unit remains the same across different channel counts.

6.2 GenStore-EM Analysis

We analyze the benefits of GenStore-EM for a 22-GB short read set where 80% of reads exactly match some subsequence in the reference genome (see Section 5 for input generation methodology), on a system with three different SSD configurations: SSD-L, SSD-M, and SSD-H.

Integration with a Software Read Mapper. Figure 10 shows the execution time of four read mapper configurations: 1) Base (Minimap2 [58]), 2) SIMD, an extension of Base with a SIMD implementation of a baseline exactly-matching read filter using 128-bit SIMD instructions, 3) GS-Ext, and 4) GS. We divide the execution time between Alignment (chaining and alignment's contribution to end-to-end execution time) and Other (file access, seeding, and exact match filtering's contribution to end-to-end execution time).



(a) Software read mapper. (b) Hardware read mapper.

Figure 9: GenStore-EM performance under different SSDs.

We make four observations from Figure 9a. First, for all SSD types, GS significantly outperforms Base and SIMD by 2.07-2.45× and 1.66-2.09×, respectively, by alleviating the cost of moving data from the SSD to the host CPU and removing the burden of finding exactly-matching reads from the rest of the system (DRAM and the processor). Second, GS-Ext provides significant performance improvements over both Base and SIMD in SSD-M and SSD-H. However, GS-Ext provides limited benefits over SIMD in SSD-L since its performance is bottlenecked by the low external I/O bandwidth. Third, even though SIMD also filters out the same number of reads and reduces the alignment time similarly to GS and GS-Ext, its average performance benefit over Base (1.19×) is quite limited compared to those of GS (2.23×) and GS-Ext (1.83×). This is because 1) both GS and GS-Ext reduce the number of memory accesses per read and convert the random memory accesses to more efficient streaming accesses, and 2) GS addresses the I/O bottlenecks due to limited external SSD bandwidth. Based on our observations, we draw two conclusions. First, GenStore-EM leads to large performance improvements in short read genomic analysis by efficiently filtering large amounts of data in storage. Second, even without in-storage processing support, the key idea of GenStore-EM can significantly improve the performance of the state-of-the-art software read mapper especially when using high-bandwidth SSDs (e.g., `SSD-M` and `SSD-H`).

Integration with a Hardware Read Mapper. Figure 9b shows the execution time of three hardware read mapper configurations: 1) Base (GenCache [43]), 2) GS-Ext, and 3) GS.[19] Integrating GenStore with existing accelerators requires no architectural changes to GenStore and the accelerators because GenStore operates directly on the original data that is stored in the SSD, before any accelerator-specific operation starts. The only factor that GenStore needs to consider is to generate its outputs (i.e., the information of unfiltered reads) in the format that the accelerator needs as its inputs. The host system is responsible for orchestrating the execution and data flows between GenStore and the accelerator.

We make two observations based on Figure 9b. First, GS significantly outperforms Base by 3.32×, 2.55×, and 1.52× on systems with `SSD-L`, `SSD-M`, and `SSD-H`, respectively. Second, GS-Ext performs significantly slower than `Base` (2.28-1.91×) on all systems since GenStore-EM requires accessing the large SSIndex data structure (Section 4.2), while GS-Ext suffers from limited SSD external bandwidth. We conclude that the in-storage processing approach in GenStore effectively addresses the I/O bottleneck, which becomes even more significant with hardware accelerators that address the computation bottleneck.

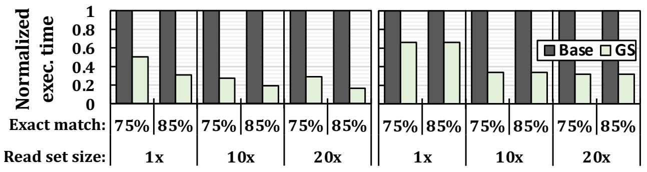
Effect of Read Set Features on Performance. We study the benefits of GenStore-EM depending on the characteristics of input read sets. To this end, we evaluate GenStore-EM while changing two key characteristics: 1) the input read set size and 2) exactly-matching read rate. These two factors affect the data movement savings (*DM_Saving*) of GenStore as governed by Equation (4):

$$DM\_Saving = \frac{Size_{Ref} + Size_{ReadSet}}{Size_{Ref} + Size_{ReadSet} \times (1 - Ratio_{Filter})}, \quad (4)$$

where $Size_{Ref}$ is the size of the reference genome and its index (e.g., 7 GB for humans [58]), $Size_{ReadSet}$ is the size of the read set, and $Ratio_{Filter}$ in GenStore-EM is the exactly-matching read rate of the read set.

Figure 10a shows the execution time of Base and GS for the baseline software read mapper (Minimap2 [58]) for input read sets with different sizes (1x, 10x, and 20x larger than the size of our default 22-GB short read set) and different exactly-matching read rates (75% and 85%) on a system with `SSD-H`. We make two observations. First, GS's performance benefit grows as input size increases (from 2.62× to 4.75×) due to larger data movement savings. Since $Size_{Ref}$ is constant, *DM_Saving* and the performance benefits of GenStore-EM increase with larger $Size_{ReadSet}$ (see Equation (4)). Second, GS's performance benefit increases with higher exactly-matching read rates (from 3.46× to 6.05× for the largest read set) because with a larger exactly-matching read rate (i.e., $Ratio_{Filter}$ in Equation (4)), data movement saving (*DM_Saving*) increases, and the time spent on mapping the unfiltered reads decreases. Mapping the unfiltered reads is the key contributor to the end-to-end execution time of GS since it has larger execution time compared to concurrently running GenStore-EM operations in the SSD. We conclude that the benefits of GenStore-EM, when integrated with the software read mapper, increases with larger read sets and with larger exactly-matching read rates.

Figure 10b shows the execution time of Base and GS for a baseline hardware read mapper (GenCache [43]). We make two observations. First, GS's performance benefit increases with larger input sizes (from 1.52× to 3.13×) due to its larger data movement reduction. Second, GS's performance benefit does *not* increase with higher exactly-matching read rates. This is because, with the hardware read mapper, the end-to-end performance of GS is dominated by the execution time of GenStore-EM's filter operations inside the SSD, which only depends on the input read set size and not on the exact-match rate. We conclude that the benefits of GenStore-EM, when integrated with the hardware read mapper, increases with larger input read set sizes.



(a) Software read mapper. **(b) Hardware read mapper.**

Figure 10: GenStore-EM performance versus input size and exact match rate.

6.3 GenStore-NM Analysis

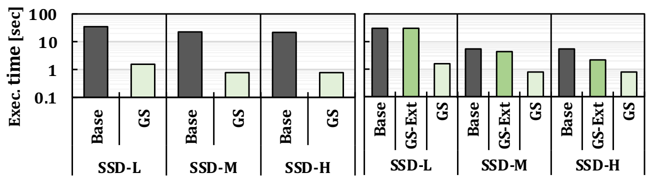
We analyze the benefits of GenStore-NM for a 12.4-GB read set (the first *No reference* use case in Table 1) with 99.65% of reads *not* aligning on a system with three different SSD configurations: `SSD-L`, `SSD-M`, and `SSD-H`.

Integration with a Software Read Mapper. Figure 11a shows the execution time of two read mapper configurations: 1) Base (Minimap2 [58]), which already incorporates the chaining filter, and 2) GS.[20] We observe that GS outperforms Base by 22.4×, 29.0×, and 27.9× on systems with `SSD-L`, `SSD-M`, and `SSD-H`, respectively.[21] The reason is that GS alleviates the cost of data movement from SSD to the processor and removes the burden of mapping a large fraction of reads from the rest of the system (DRAM and the processor).

[19]We do not show the execution breakdown of mapping in Figure 9b since we cannot obtain the execution time breakdown for the hardware read mapper from [43].

[20]Unlike Figure 9a, we do not divide the execution times of `Alignment` and `Other` in Figure 11a since the execution time of GS is dominated by the filter operations of GenStore-NM. This is because, in this case, a large fraction (e.g., 99.65% in our evaluated dataset) of reads get filtered in the SSD and the execution time of mapping the unfiltered reads is very small.

[21]In this case, GS provides larger benefits for systems with `SSD-M` and `SSD-H` since these SSDs have larger internal bandwidth compared to `SSD-L`.

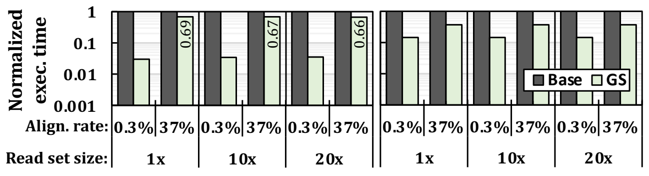


(a) Software read mapper. **(b) Hardware read mapper.**

Figure 11: GenStore-NM performance under different SSDs.

Integration with a Hardware Read Mapper. Figure 11b shows the execution time of three hardware read mapper configurations: 1) Base (Darwin [39]), 2) GS-Ext, and 3) GS. We make two observations based on Figure 11b. First, GS significantly outperforms Base by 19.2×, 6.86×, and 6.85× on systems with SSD-L, SSD-M, and SSD-H, respectively. The reason is that GS alleviates the cost of data movement from SSD to the accelerator and removes the burden of read mapping for the filtered reads from the rest of the system (DRAM and the accelerator). Second, GS-Ext does not provide significant performance benefits compared to Base in systems with SSD-L and SSD-M since the execution time of GS-Ext is dominated by the I/O overhead of bringing reads from SSD to the accelerator. GS-Ext performs 2.50× faster than Base in systems with SSD-H since the I/O bottlenecks are partially alleviated with SSD-H. However, the benefits of GS-Ext are limited compared to GS since GS significantly alleviates the I/O overhead of bringing reads from SSD to the accelerator with all three SSD configurations.

Effect of Read Set Features on Performance. We study how the benefits of GenStore-NM vary depending on the characteristics of input read sets. To this end, we evaluate GenStore-NM while changing two key characteristics: 1) the input read set size and 2) alignment rates (the fraction of reads in the read set that align to the reference genome). We use input sets with different sizes (1×, 10×, and 20× larger than the size of our default 12.5GB read set) with different alignment rates (0.3% and 37%, corresponding to the first and second *No reference* use cases in Table 1), on a system with SSD-H. Figure 12 shows the execution time of Base and GS for the baseline software read mapper [58] (Figure 12a) and the baseline hardware read mapper [39] (Figure 12b).



(a) Software read mapper. **(b) Hardware read mapper.**

Figure 12: GenStore-NM performance versus input size and read alignment rate.

We make two main observations from Figure 12. First, for both hardware and software read mappers, GS's performance benefits vary little as the input size changes. The reason is that $Size_{Ref}$ is very small (14.6 MB) for this experiment. Therefore, data movement saving (*DM_Saving* in Equation (4)) mainly correlates with alignment rate (i.e., $1 - Ratio_{Filter}$). Second, for both software and hardware read mappers, GS's performance benefit increases as the ratio of non-aligning reads increases (i.e., as alignment rate reduces), because with higher values of $Ratio_{Filter}$, both 1) *DM_Saving* increases and 2) the time spent on mapping unfiltered reads decreases. We conclude the GenStore-NM provides high performance benefits to both software and hardware read mappers with different input sizes and its benefits increase with lower alignment rates.

6.4 Energy Analysis

To demonstrate the energy benefits of different GenStore modes, we obtain the energy of the host processor, the host-side DRAM, the DRAM inside the SSD, the communication between the SSD and the host, active and idle energy of the SSD, and the energy of the logic units used in GenStore (Section 6.1). We calculate the energy of each component based on its idle and dynamic power consumption and its execution time. We observe that by filtering out large amounts of data, GenStore reduces the end-to-end energy consumption of read mapping in all of our evaluations compared to Base (Minimap2 [58]). We measure the energy consumption for experiments on all SSD configurations. By filtering exact matches in a short read set with 80% exactly-matching read rate (see Section 5), GenStore-EM reduces the energy consumption by on average (up to) 3.92× (3.97×) across all storage configurations. By filtering non-matching reads in a long long read set with a read alignment rate of 0.35% (the first *No reference* use case in Table 1), GenStore-NM reduces the energy consumption by on average (up to) 27.17× (29.25×) across all storage configurations.

7 RELATED WORK

To our knowledge, GenStore is the *first* in-storage system designed for accelerating genome sequence analysis. GenStore works with both short and long reads, and different degrees of genetic variation between the compared genomes.

Accelerating Read Mapping. There exists a large body of work on accelerating read mapping, which tend to follow two general directions: 1) non-filtering and 2) filtering approaches [35]. Neither of these two approaches are designed for processing in storage. Non-filtering approaches accelerate one or more non-filtering steps of read mapping (e.g., seeding and approximate string matching) using hardware accelerators. Examples of these accelerators include processing-in-memory architectures [40, 61, 65–69], ASICs [39, 62, 70], GPUs [71–73, 78–84, 188, 189], and FPGAs [63, 64, 74–77, 85–93]. Filtering approaches accelerate the pre-alignment filtering step of read mapping. These works provide highly-parallel read filtering heuristics that quickly eliminate dissimilar sequences before invoking computationally-expensive alignment algorithms. Examples of these accelerators include processing-near-memory architectures [43, 94, 95, 97], FPGAs [41, 42, 48, 49, 98], GPUs [41, 96, 98], or traditional CPU-based acceleration [37, 38, 41].

In contrast to GenStore, prior works on read mapping acceleration suffer from two key issues. First, they all *still* need to access the genomic data that is initially stored in the storage device, which incurs time and energy costs due to data movement overhead from the storage device to the main memory and later to the CPU cores or accelerators. Second, most prior works support processing *only* one type of sequencing reads [37–39, 42, 48, 49, 62, 64, 65, 68, 69, 71–73, 77, 82–84, 86, 90, 91, 96, 98], which limits their applicability. Compared to existing filtering approaches, our work 1) is the first to identify the I/O inefficiencies that are not addressed by existing techniques and 2) introduces new filtering techniques that enable efficient in-storage processing for both short and long reads.

In-Storage Processing Systems. Prior works explore in-storage processing in the form of application-specific accelerators [143, 190–196], general-purpose processing inside storage devices [195–201], SSDs closely integrated with FPGAs [142, 202–206], or SSDs closely integrated with GPUs [207]. Several works propose techniques for general pattern matching in storage [208, 209] *without* a specific focus on read mapping *nor* support for different sequencing read types and data structures. None of these works perform in-storage filtering for read mapping to accelerate genome sequence analysis.

8 DISCUSSION

As sequencing technologies develop in the future, we expect that GenStore will still play an important role in genomic sequence analysis. We witness three major trends in sequencing technologies. First, the length and the accuracy of reads are expected to increase [35, 53, 54, 210, 211]. Second, short reads will continue to be widely used due to their *very* high accuracy and low cost [26, 212, 213]. Third, DNA sequencing machines are increasingly adopting data processing capabilities to perform *both* sequencing and genomic analysis within the same machine [31, 34, 214–217].

Even with increases in long read accuracy (*trend 1*), GenStore-NM would continue to filter large numbers of reads that do *not* align to a reference genome due to genetic differences between the compared genomes [155, 218–222]. Given that accurate short reads are essential for many processes in genome analysis (*trend 2*), e.g., for polishing [46, 223, 224] and validating [225] long read sequences, GenStore-EM can continue to accelerate short read mapping by filtering out exactly-matching reads. With the push to provide DNA sequencing and preliminary genetic analysis on more portable integrated devices with limited compute resources and available DRAM capacity [34, 35, 214, 226] (*trend 3*), there will be a growing demand for data movement-minimizing systems like GenStore to quickly filter out a large fraction of reads at low cost and high efficiency [94, 100, 214, 217, 227, 228].

With higher availability of portable genome sequencers, genomic analyses will be performed more routinely [1–9, 35, 99, 155, 210, 228, 229] and will need to provide much faster results [35, 94]. In-storage processing is crucial for meeting the huge demands for faster analyses that scale well to large numbers of genomic samples. Examples of these analyses include gene detection [230], alignment of reads from rapidly mutating organisms such as SARS-CoV-2 [9, 100], and studies of as-of-yet undiscovered microbes [155, 218, 219, 229]. In these analyses, GenStore-NM can effectively filter out >99.7% [219, 230], ~36.1% [100], and ~47.3% [155, 218, 219] of the reads, respectively, thereby greatly improving the end-to-end throughput and energy efficiency of genome sequence analysis. We hope that our proposed techniques provide a foundation for future efforts to accelerate genome analysis.

9 CONCLUSION

We propose GenStore, a new in-storage processing system for genome sequence analysis. GenStore can be integrated with both hardware and software read mappers to improve the end-to-end performance of both short and long read mapping. We address the challenges of in-storage processing for genomic read filtering via new hardware/software co-designed techniques and develop new in-storage filtering accelerators for both short and long reads. Our evaluations show that GenStore provides large performance and energy improvements when integrated into the state-of-the-art software and hardware read mappers. We hope that GenStore inspires other in-storage processing and storage system design ideas for genome sequence analysis. A promising avenue for future work is to enable the integration of GenStore-like accelerators into sequencing devices for extreme scalability and efficiency.

ACKNOWLEDGMENTS

We thank the anonymous reviewers of MICRO 2021 and ASPLOS 2022 for feedback. We thank the SAFARI group members for feedback and the stimulating intellectual environment. We acknowledge the generous gifts and support provided by our industrial partners: Google, Huawei, Intel, Microsoft, VMware, and the Semiconductor Research Corporation. Jisung Park was in part supported by the National Research Foundation (NRF) of Korea (NRF-2020R1A6A3A03040573).

REFERENCES

[1] Michelle M Clark, Amber Hildreth, Sergey Batalov, Yan Ding, Shimul Chowdhury, et al. Diagnosis of Genetic Diseases in Seriously Ill Children by Rapid Whole-genome Sequencing and Automated Phenotyping and Interpretation. *Science Translational Medicine*, 2019.
[2] Lauge Farnaes, Amber Hildreth, Nathaly M Sweeney, Michelle M Clark, Shimul Chowdhury, et al. Rapid Whole-genome Sequencing Decreases Infant Morbidity and Cost of Hospitalization. *NPJ Genomic Medicine*, 2018.
[3] Nathaly M Sweeney, Shareef A Nahas, Shimul Chowdhury, Sergey Batalov, Michelle Clark, et al. Rapid Whole Genome Sequencing Impacts Care and Resource Utilization in Infants with Congenital Heart Disease. *NPJ Genomic Medicine*, 2021.
[4] Can Alkan, Jeffrey M Kidd, Tomas Marques-Bonet, Gozde Aksay, Francesca Antonacci, et al. Personalized Copy Number and Segmental Duplication Maps Using Next-Generation Sequencing. *Nature Genetics*, 2009.
[5] Mauricio Flores, Gustavo Glusman, Kristin Brogaard, Nathan D Price, and Leroy Hood. P4 Medicine: How Systems Medicine Will Transform the Healthcare Sector and Society. *Personalized Medicine*, 2013.
[6] Geoffrey S Ginsburg and Huntington F Willard. Genomic and Personalized Medicine: Foundations and Applications. *Translational Research*, 2009.
[7] Lynda Chin, Jannik N Andersen, and P Andrew Futreal. Cancer Genomics: From Discovery Science to Personalized Medicine. *Nature Medicine*, 2011.
[8] Euan A Ashley. Towards Precision Medicine. *Nature Reviews Genetics*, 2016.
[9] Joshua S Bloom, Laila Sathe, Chetan Munugala, Eric M Jones, Molly Gasperini, et al. Massively Scaled-up Testing for SARS-CoV-2 RNA via Next-generation Sequencing of Pooled and Barcoded Nasal and Saliva Samples. *Nature Biomedical Engineering*, 2021.
[10] Ramesh Yelagandula, Aleksandr Bykov, Alexander Vogt, Robert Heinen, Ezgi Özkan, et al. Multiplexed Detection of SARS-CoV-2 and Other Respiratory Infections in High Throughput by SARSeq. *Nature Communications*, 2021.
[11] Vien Thi Minh Le and Binh An Diep. Selected Insights from Application of Whole Genome Sequencing for Outbreak Investigations. *Current Opinion in Critical Care*, 2013.
[12] Vlad Nikolayevskyy, Katharina Kranzer, Stefan Niemann, and Francis Drobniewski. Whole Genome Sequencing of Mycobacterium Tuberculosis for Detection of Recent Transmission and Tracing Outbreaks: A Systematic Review. *Tuberculosis*, 2016.
[13] Shaofu Qiu, Peng Li, Hongbo Liu, Yong Wang, Nan Liu, et al. Whole-genome Sequencing for Tracing the Transmission Link between Two ARD Outbreaks Caused by A Novel HAdV Serotype 7 Variant, China. *Scientific Reports*, 2015.
[14] Carol A Gilchrist, Stephen D Turner, Margaret F Riley, William A Petri, and Erik L Hewlett. Whole-genome Sequencing in Outbreak Analysis. *Clinical Microbiology Reviews*, 2015.
[15] Sean Hoban, Joanna L Kelley, Katie E Lotterhos, Michael F Antolin, Gideon Bradburd, et al. Finding the Genomic Basis of Local Adaptation: Pitfalls, Practical Solutions, and Future Directions. *The American Naturalist*, 2016.

[16] J Romiguier, Philippe Gayral, Marion Ballenghien, Arnaud Bernard, Vincent Cahais, et al. Comparative Population Genomics in Animals Uncovers the Determinants of Genetic Diversity. *Nature*, 2014.
[17] Hans Ellegren and Nicolas Galtier. Determinants of Genetic Diversity. *Nature Reviews Genetics*, 2016.
[18] Ana Prohaska, Fernando Racimo, Andrew J Schork, Martin Sikora, Aaron J Stern, et al. Human Disease Variation in the Light of Population Genomics. *Cell*, 2019.
[19] Hans Ellegren. Genome Sequencing and Population Genomics in Non-Model Organisms. *Trends in Ecology & Evolution*, 2014.
[20] Javier Prado-Martinez, Peter H. Sudmant, Jeffrey M. Kidd, Heng Li, Joanna L. Kelley, et al. Great Ape Genetic Diversity and Population History. *Nature*, 2013.
[21] Ana Prohaska, Fernando Racimo, Andrew J Schork, Martin Sikora, Aaron J Stern, et al. Human Disease Variation in the Light of Population Genomics. *Cell*, 2019.
[22] Jason A Reuter, Damek V Spacek, and Michael P Snyder. High-Throughput Sequencing Technologies. *Molecular Cell*, 2015.
[23] Erwin L van Dijk, Hélène Auger, Yan Jaszczyszyn, and Claude Thermes. Ten Years of Next-Generation Sequencing Technology. *Trends in Genetics*, 2014.
[24] Travis C Glenn. Field Guide to Next-Generation DNA Sequencers. *Molecular Ecology Resources*, 2011.
[25] Sara Goodwin, John D McPherson, and W Richard McCombie. Coming of Age: Ten Years of Next-generation Sequencing Technologies. *Nature Reviews Genetics*, 2016.
[26] Michael A Quail, Miriam Smith, Paul Coupland, Thomas D Otto, Simon R Harris, et al. A Tale of Three Next Generation Sequencing Platforms: Comparison of Ion Torrent, Pacific Biosciences and Illumina MiSeq Sequencers. *BMC Genomics*, 2012.
[27] Mehdi Kchouk, Jean-Francois Gibrat, and Mourad Elloumi. Generations of Sequencing Technologies: from First to Next Generation. *Biology and Medicine*, 2017.
[28] Franziska Pfeiffer, Carsten Gröber, Michael Blank, Kristian Händler, Marc Beyer, et al. Systematic Evaluation of Error Rates and Causes in Short Samples in Next-generation Sequencing. *Scientific Reports*, 2018.
[29] Shanika L Amarasinghe, Shian Su, Xueyi Dong, Luke Zappia, Matthew E Ritchie, and Quentin Gouil. Opportunities and Challenges in Long-read Sequencing Data Analysis. *Genome Biology*, 2020.
[30] Damla Senol Cali, Jeremie S Kim, Saugata Ghose, Can Alkan, and Onur Mutlu. Nanopore Sequencing Technology and Tools for Genome Assembly: Computational Analysis of the Current State, Bottlenecks and Future Directions. *Briefings in Bioinformatics*, 2018.
[31] Simon Ardui, Adam Ameur, Joris R Vermeesch, and Matthew S Hestand. Single Molecule Real-Time (SMRT) Sequencing Comes of Age: Applications and Utilities for Medical Diagnostics. *Nucleic Acids Research*, 2018.
[32] Jason L Weirather, Mariateresa de Cesare, Yunhao Wang, Paolo Piazza, Vittorio Sebastiano, et al. Comprehensive Comparison of Pacific Biosciences and Oxford Nanopore Technologies and Their Applications to Transcriptome Analysis. *F1000Research*, 2017.
[33] Erwin L van Dijk, Yan Jaszczyszyn, Delphine Naquin, and Claude Thermes. The Third Revolution in Sequencing Technology. *Trends in Genetics*, 2018.
[34] Yunhao Wang, Yue Zhao, Audrey Bollas, Yuru Wang, and Kin Fai Au. Nanopore Sequencing Technology, Bioinformatics and Applications. *Nature Biotechnology*, 2021.
[35] Mohammed Alser, Zülal Bingöl, Damla Senol Cali, Jeremie Kim, Saugata Ghose, et al. Accelerating Genome Analysis: A Primer on An Ongoing Journey. *IEEE Micro*, 2020.
[36] Mohammed Alser, Jeremy Rotman, Kodi Taraszka, Huwenbo Shi, Pelin Icer Baykal, et al. Technology Dictates Algorithms: Recent Developments in Read Alignment. *Genome Biology*, 2021.
[37] Hongyi Xin, Donghyuk Lee, Farhad Hormozdiari, Samihan Yedkar, Onur Mutlu, and Can Alkan. Accelerating Read Mapping with FastHASH. *BMC Genomics*, 2013.
[38] Hongyi Xin, John Greth, John Emmons, Gennady Pekhimenko, Carl Kingsford, et al. Shifted Hamming Distance: A Fast and Accurate SIMD-friendly Filter to Accelerate Alignment Verification in Read Mapping. *Bioinformatics*, 2015.
[39] Yatish Turakhia, Gill Bejerano, and William J Dally. Darwin: A Genomics Co-processor Provides up to 15,000 x Acceleration on Long Read Assembly. In *ASPLOS*, 2018.
[40] Damla Senol Cali, Gupreet Kalsi, Zulal Bingöl, Lavanya Subramanian, Can Firtina, et al. GenASM: A High-Performance, Low-Power Approximate String Matching Acceleration Framework for Genome Sequence Analysis. In *MICRO*, 2020.
[41] Mohammed Alser, Taha Shahroodi, Juan Gómez-Luna, Can Alkan, and Onur Mutlu. SneakySnake: A Fast and Accurate Universal Genome Pre-alignment Filter for CPUs, GPUs and FPGAs. *Bioinformatics*, 2020.
[42] Mohammed Alser, Hasan Hassan, Hongyi Xin, Oğuz Ergin, Onur Mutlu, and Can Alkan. GateKeeper: A New Hardware Architecture for Accelerating Pre-alignment in DNA Short Read Mapping. *Bioinformatics*, 2017.
[43] Anirban Nag, CN Ramachandra, Rajeev Balasubramonian, Ryan Stutsman, Edouard Giacomin, et al. GenCache: Leveraging In-cache Operators for Efficient Sequence Alignment. In *MICRO*, 2019.
[44] Jeremie S Kim, Can Firtina, Meryem Banu Cavlak, Damla Senol Cali, Mohammed Alser, et al. AirLift: A Fast and Comprehensive Technique for Remapping Alignments between Reference Genomes. *arXiv*, 2019.
[45] Jeremie S Kim, Damla Senol Cali, Hongyi Xin, Donghyuk Lee, Saugata Ghose, et al. GRIM-Filter: Fast Seed Location Filtering in DNA Read Mapping Using Processing-in-memory Technologies. *BMC Genomics*, 2018.
[46] Can Firtina, Jeremie S Kim, Mohammed Alser, Damla Senol Cali, A Ercument Cicek, et al. Apollo: A Sequencing-technology-independent, Scalable and Accurate Assembly Polishing Algorithm. *Bioinformatics*, 2020.
[47] Martin Šošić and Mile Šikić. Edlib: A C/C++ Library for Fast, Exact Sequence Alignment Using Edit Distance. *Bioinformatics*, 2017.
[48] Mohammed Alser, Onur Mutlu, and Can Alkan. MAGNET: Understanding and Improving the Accuracy of Genome Pre-alignment Filtering. *arXiv*, 2017.
[49] Mohammed Alser, Hasan Hassan, Akash Kumar, Onur Mutlu, and Can Alkan. Shouji: A Fast and Efficient Pre-alignment Filter for Sequence Alignment. *Bioinformatics*, 2019.
[50] Saul B Needleman and Christian D Wunsch. A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins. *Journal of Molecular Biology*, 1970.
[51] Temple F Smith, Michael S Waterman, et al. Identification of Common Molecular Subsequences. *Journal of Molecular Biology*, 1981.
[52] Osamu Gotoh. An Improved Algorithm for Matching Biological Sequences. *Journal of Molecular Biology*, 1982.
[53] Shawn E Levy and Richard M Myers. Advancements in Next-generation Sequencing. *Annual Review of Genomics and Human Genetics*, 2016.
[54] Taishan Hu, Nilesh Chitnis, Dimitri Monos, and Anh Dinh. Next-Generation Sequencing Technologies: An Overview. *Human Immunology*, 2021.
[55] 1000 Genomes Project Consortium et al. A Global Reference for Human Genetic Variation. *Nature*, 2015.
[56] Zheng Zhang, Scott Schwartz, Lukas Wagner, and Webb Miller. A Greedy Algorithm for Aligning DNA Sequences. *Journal of Computational Biology*, 2000.
[57] Guy St C Slater and Ewan Birney. Automated Generation of Heuristics for Biological Sequence Comparison. *BMC Bioinformatics*, 2005.
[58] Heng Li. Minimap2: Pairwise Alignment for Nucleotide Sequences. *Bioinformatics*, 2018.
[59] Gene Myers. A Fast Bit-vector Algorithm for Approximate String Matching Based on Dynamic Programming. *JACM*, 1999.
[60] Santiago Marco-Sola, Juan Carlos Moure, Miquel Moreto, and Antonio Espinosa. Fast Gap-affine Pairwise Alignment Using the Wavefront Algorithm. *Bioinformatics*, 2021.
[61] Wenqin Huangfu, Shuangchen Li, Xing Hu, and Yuan Xie. RADAR: A 3D-ReRAM based DNA Alignment Accelerator Architecture. In *DAC*, 2018.
[62] Daichi Fujiki, Arun Subramaniyan, Tianjun Zhang, Yu Zeng, Reetuparna Das, et al. Genax: A Genome Sequencing Accelerator. In *ISCA*, 2018.
[63] Daichi Fujiki, Shunhao Wu, Nathan Ozog, Kush Goliya, David Blaauw, et al. SeedEx: A Genome Sequencing Accelerator for Optimal Alignments in Subminimal Space. In *MICRO*, 2020.
[64] Subho Sankar Banerjee, Mohamed El-Hadedy, Jong Bin Lim, Zbigniew T Kalbarczyk, Deming Chen, et al. ASAP: Accelerated Short-read Alignment on Programmable Hardware. *IEEE Transactions on Computers*, 2019.
[65] S Karen Khatamifard, Zamshed Chowdhury, Nakul Pande, Meisam Razaviyayn, Chris H Kim, and Ulya R Karpuzcu. GeNVoM: Read Mapping Near Non-Volatile Memory. *TCBB*, 2021.
[66] Saransh Gupta, Mohsen Imani, Behnam Khaleghi, Venkatesh Kumar, and Tajana Rosing. RAPID: A ReRAM Processing In-memory Architecture for DNA Sequence Alignment. In *ISLPED*, 2019.
[67] Xue-Qi Li, Guang-Ming Tan, and Ning-Hui Sun. PIM-Align: A Processing-in-Memory Architecture for FM-Index Search Algorithm. *Journal of Computer Science and Technology*, 2021.
[68] Shaahin Angizi, Jiao Sun, Wei Zhang, and Deliang Fan. Aligns: A Processing-in-memory Accelerator for DNA Short Read Alignment Leveraging SOT-MRAM. In *DAC*, 2019.
[69] Farzaneh Zokaee, Hamid R Zarandi, and Lei Jiang. Aligner: A Process-in-memory Architecture for Short Read Alignment in ReRAMs. *IEEE Computer Architecture Letters*, 2018.
[70] Advait Madhavan, Timothy Sherwood, and Dmitri Strukov. Race Logic: A Hardware Acceleration for Dynamic Programming Algorithms. *ACM SIGARCH Computer Architecture News*, 2014.
[71] Haoyu Cheng, Yong Zhang, and Yun Xu. Bitmapper2: A GPU-accelerated All-mapper Based on The Sparse Q-gram Index. *TCBB*, 2018.
[72] Ernst Joachim Houtgast, Vlad-Mihai Sima, Koen Bertels, and Zaid Al-Ars. Hardware Acceleration of BWA-MEM Genomic Short Read Mapping for Longer Read Lengths. *Computational Biology and Chemistry*, 2018.

[73] Ernst Joachim Houtgast, VladMihai Sima, Koen Bertels, and Zaid AlArs. An Efficient GPU-accelerated Implementation of Genomic Short Read Mapping with BWA-MEM. *ACM SIGARCH Computer Architecture News*, 2017.

[74] Amit Goyal, Hyuk Jung Kwon, Kichan Lee, Reena Garg, Seon Young Yun, et al. Ultra-fast Next Generation Human Genome Sequencing Data Processing Using DRAGENTM Bio-IT Processor for Precision Medicine. *Open Journal of Genetics*, 2017.

[75] Yu-Ting Chen, Jason Cong, Zhenman Fang, Jie Lei, and Peng Wei. When Spark Meets FPGAs: A Case Study for Next-Generation DNA Sequencing Acceleration. In *USENIX HotCloud*, 2016.

[76] Peng Chen, Chao Wang, Xi Li, and Xuehai Zhou. Accelerating the Next Generation Long Read Mapping with the FPGA-based System. *TCBB*, 2014.

[77] Yen-Lung Chen, Bo-Yi Chang, Chia-Hsiang Yang, and Tzi-Dar Chiueh. A High-Throughput FPGA Accelerator for Short-Read Mapping of the Whole Human Genome. *IEEE TPDS*, 2021.

[78] Alberto Zeni, Giulia Guidi, Marquita Ellis, Nan Ding, Marco D Santambrogio, et al. Logan: High-performance GPU-based X-drop Long-read Alignment. In *IPDPS*, 2020.

[79] Nauman Ahmed, Jonathan Lévy, Shanshan Ren, Hamid Mushtaq, Koen Bertels, and Zaid Al-Ars. GASAL2: A GPU Accelerated Sequence Alignment Library for High-Throughput NGS Data. *BMC Bioinformatics*, 2019.

[80] Takahiro Nishimura, Jacir L Bordim, Yasuaki Ito, and Koji Nakano. Accelerating the Smith-waterman Algorithm Using Bitwise Parallel Bulk Computation Technique on GPU. In *IPDPSW)*, 2017.

[81] Edans Flavius de Oliveira Sandes, Guillermo Miranda, Xavier Martorell, Eduard Ayguade, George Teodoro, and Alba Cristina Magalhaes Melo. CUDAlign 4.0: Incremental Speculative Traceback for Exact Chromosome-wide Alignment in GPU Clusters. *IEEE TPDS*, 2016.

[82] Yongchao Liu and Bertil Schmidt. GSWABE: Faster GPU-accelerated Sequence Alignment with Optimal Alignment Retrieval for Short DNA Sequences. *Concurrency and Computation: Practice and Experience*, 2015.

[83] Yongchao Liu, Adrianto Wirawan, and Bertil Schmidt. CUDASW++ 3.0: Accelerating Smith-Waterman Protein Database Search by Coupling CPU and GPU SIMD Instructions. *BMC Bioinformatics*, 2013.

[84] Richard Wilton, Tamas Budavari, Ben Langmead, Sarah J Wheelan, Steven L Salzberg, and Alexander S Szalay. Arioc: High-throughput Read Alignment with GPU-accelerated Exploration of The Seed-and-extend Search Space. *PeerJ*, 2015.

[85] Xia Fei, Zou Dan, Lu Lina, Man Xin, and Zhang Chunlei. FPGASW: Accelerating Large-scale Smith–Waterman Sequence Alignment Application with Backtracking on FPGA Linear Systolic Array. *Interdisciplinary Sciences: Computational Life Sciences*, 2018.

[86] Hasitha Muthumala Waidyasooriya and Masanori Hariyama. Hardware-acceleration of Short-read Alignment Based on the Burrows-wheeler Transform. *TPDS*, 2015.

[87] Yu-Ting Chen, Jason Cong, Jie Lei, and Peng Wei. A Novel High-throughput Acceleration Engine for Read Alignment. In *FCCM*, 2015.

[88] Enzo Rucci, Carlos Garcia, Guillermo Botella, Armando De Giusti, Marcelo Naiouf, and Manuel Prieto-Matias. SWIFOLD: Smith-Waterman Implementation on FPGA with OpenCL for Long DNA Sequences. *BMC Systems Biology*, 2018.

[89] Abbas Haghi, Santiago Marco-Sola, Lluc Alvarez, Dionysios Diamantopoulos, Christoph Hagleitner, and Miquel Moreto. An FPGA Accelerator of the Wavefront Algorithm for Genomics Pairwise Alignment. In *FPL*, 2021.

[90] Luyi Li, Jun Lin, and Zhongfeng Wang. PipeBSW: A Two-Stage Pipeline Structure for Banded Smith-Waterman Algorithm on FPGA. In *ISVLSI*, 2021.

[91] Ham, Tae Jun and Bruns-Smith, David and Sweeney, Brendan and Lee, Yejin and Seo, Seong Hoon and Song, U Gyeong and Oh, Young H and Asanovic, Krste and Lee, Jae W and Wills, Lisa Wu. Genesis: A Hardware Acceleration Framework for Genomic Data Analysis. In *ISCA*, 2020.

[92] Tae Jun Ham, Yejin Lee, Seong Hoon Seo, U Gyeong Song, Jae W Lee, et al. Accelerating Genomic Data Analytics With Composable Hardware Acceleration Framework. *IEEE Micro*, 2021.

[93] Lisa Wu, David Bruns-Smith, Frank A Nothaft, Qijing Huang, Sagar Karandikar, et al. FPGA Accelerated Indel Realignment in the Cloud. In *HPCA*, 2019.

[94] Gagandeep Singh, Mohammed Alser, Damla Senol Cali, Dionysios Diamantopoulos, Juan Gómez-Luna, et al. FPGA-based Near-Memory Acceleration of Modern Data-Intensive Applications. *IEEE Micro*, 2021.

[95] Jung-Sik Kim, Chi Sung Oh, Hocheol Lee, Donghyuk Lee, Hyong-Ryol Hwang, et al. A 1.2 v 12.8 gb/s 2gb Mobile Wide-i/o Dram with 4× 128 i/os Using TSV-based Stacking. In *ISSCC*, 2011.

[96] Zülal Bingöl, Mohammed Alser, Onur Mutlu, Ozcan Ozturk, and Can Alkan. Gatekeeper-gpu: Fast and accurate pre-alignment filtering in short read mapping. *IPDPSW*, 2021.

[97] Fazal Hameed, Asif Ali Khan, and Jeronimo Castrillon. ALPHA: A Novel Algorithm-Hardware Co-design for Accelerating DNA Seed Location Filtering. *ITETC*, 2021.

[98] Licheng Guo, Jason Lau, Zhenyuan Ruan, Peng Wei, and Jason Cong. Hardware Acceleration of Long Read Pairwise Overlapping in Genome Sequencing: A Race between FPGA and GPU. In *FCCM*, 2019.

[99] UK10K consortium et al. The UK10K Project Identifies Rare Variants in Health and Disease. *Nature*, 2015.

[100] Rahul C Bhoyar, Abhinav Jain, Paras Sehgal, Mohit Kumar Divakar, Disha Sharma, et al. High Throughput Detection and Genetic Epidemiology of SARS-CoV-2 Using COVIDSeq Next-generation Sequencing. *PloS One*, 2021.

[101] Yoongu Kim, Weikun Yang, and Onur Mutlu. Ramulator: A Fast and Extensible DRAM Simulator. *CAL*, 2015.

[102] Arash Tavakkol, Juan Gómez-Luna, Mohammad Sadrosadati, Saugata Ghose, and Onur Mutlu. MQSim: A Framework for Enabling Realistic Studies of Modern Multi-queue SSD Devices. In *FAST*, 2018.

[103] Fraz Syed, Haiying Grunenwald, and Nicholas Caruccio. Next-generation Sequencing Library Preparation: Simultaneous Fragmentation and Tagging Using in Vitro Transposition. *Nature Methods*, 2009.

[104] Jay Shendure and Hanlee Ji. Next-generation DNA Sequencing. *Nature Biotechnology*, 2008.

[105] Hasindu Gamaarachchi, Hiruna Samarakoon, Sasha P Jenner, James M Ferguson, Timothy G Amos, et al. Fast Nanopore Sequencing Data Analysis with SLOW5. *Nature Biotechnology*, 2022.

[106] Oxford Nanopore Technologies. MinION Mk1B IT Requirements. https://community.nanoporetech.com/requirements_documents/minion-it-reqs.pdf, 2021.

[107] Rasko Leinonen, Hideaki Sugawara, Martin Shumway, and International Nucleotide Sequence Database Collaboration. The Sequence Read Archive. *Nucleic Acids Research*, 2010.

[108] Stephan Köstlbacher, Astrid Collingro, Tamara Halter, Frederik Schulz, Sean P Jungbluth, and Matthias Horn. Pangenomics Reveals Alternative Environmental Lifestyles among Chlamydiae. *Nature Communications*, 2021.

[109] Lucy van Dorp, Mislav Acman, Damien Richard, Liam P Shaw, Charlotte E Ford, et al. Emergence of Genomic Diversity and Recurrent Mutations in SARS-CoV-2. *Infection, Genetics and Evolution*, 2020.

[110] Nathan LaPierre, Mohammed Alser, Eleazar Eskin, David Koslicki, and Serghei Mangul. Metalign: Efficient Alignment-based Metagenomic Profiling Via Containment Min Hash. *Genome Biology*, 2020.

[111] F Meyer, A Fritz, Z-L Deng, D Koslicki, A Gurevich, et al. Critical Assessment of Metagenome Interpretation-the second round of challenges. *bioRxiv*, 2021.

[112] Michael M Khayat, Sayed Mohammad Ebrahim Sahraeian, Samantha Zarate, Andrew Carroll, Huixiao Hong, et al. Hidden Biases in Germline Structural Variant Detection. *Genome Biology*, 2021.

[113] Ruibang Luo, Yiu-Lun Wong, Wai-Chun Law, Lap-Kei Lee, Jeanno Cheung, et al. BALSA: Integrated Secondary Analysis for Whole-genome and Whole-exome Sequencing, Accelerated by GPU. *PeerJ*, 2014.

[114] Hongyi Xin, Sunny Nahar, Richard Zhu, John Emmons, Gennady Pekhimenko, et al. Optimal Seed Solver: Optimizing Seed Selection in Read Mapping. *Bioinformatics*, 2016.

[115] Stephen F Altschul, Warren Gish, Webb Miller, Eugene W Myers, and David J Lipman. Basic Local Alignment Search Tool. *Journal of Molecular Biology*, 1990.

[116] Derrick E Wood, Jennifer Lu, and Ben Langmead. Improved Metagenomic Analysis with Kraken 2. *Genome Biology*, 2019.

[117] Saul Schleimer, Daniel S Wilkerson, and Alex Aiken. Winnowing: Local Algorithms for Document Fingerprinting. In *ACM SIGMOD*, 2003.

[118] Michael Roberts, Wayne Hayes, Brian R Hunt, Stephen M Mount, and James A Yorke. Reducing Storage Requirements for Biological Sequence Comparison. *Bioinformatics*, 2004.

[119] Guillaume Marçais, David Pellow, Daniel Bork, Yaron Orenstein, Ron Shamir, and Carl Kingsford. Improving the Performance of Minimizers and Winnowing Schemes. *Bioinformatics*, 2017.

[120] Heng Li. Minimap and Miniasm: Fast Mapping and De Novo Assembly for Noisy Long Sequences. *Bioinformatics*, 2016.

[121] Samsung. Samsung SSD 860 PRO. https://www.samsung.com/semiconductor/minisite/ssd/product/consumer/860pro/, 2018.

[122] Jisung Park, Myungsuk Kim, Sungjin Lee, and Jihong Kim. Improving I/O Performance of Large-page Flash Storage Systems Using Subpage-parallel Reads. In *NVMSA*, 2018.

[123] Myungsuk Kim, Jaehoon Lee, Sungjin Lee, Jisung Park, Youngsun Song, and Jihong Kim. Improving Performance and Lifetime of Large-page NAND Storages Using Erase-free Subpage Programming. In *DAC*, 2017.

[124] Intel. Intel SSD DC S4500 Series. https://ark.intel.com/content/www/us/en/ark/products/120521/intel-ssd-dc-s4500-series-480gb-2-5in-sata-6gbs-3d1-tlc.html, 2017.

[125] Samsung. Samsung SSD PM1735. https://www.samsung.com/semiconductor/ssd/enterprise-ssd/MZPLJ3T2HBJR-00007/, 2020.

[126] AnandTech. New Enterprise SSD Controllers. https://www.anandtech.com/show/16275/new-enterprise-ssd-controllers-from-silicon-motion-phison-fadu.

[127] Arash Tavakkol, Mohammad Sadrosadati, Saugata Ghose, Jeremie Kim, Yixin Luo, et al. FLIN: Enabling Fairness and Enhancing Performance in Modern NVMe Solid State Drives. In *ISCA*, 2018.

[128] Myungsuk Kim, Jisung Park, Genhee Cho, Yoona Kim, Lois Orosa, et al. Evanesco: Architectural Support for Efficient Data Sanitization in Modern Flash-Based Storage Systems. In *ASPLOS*, 2020.

[129] Yu Cai, Saugata Ghose, Erich F Haratsch, Yixin Luo, and Onur Mutlu. Error Characterization, Mitigation, and Recovery in Flash-Memory-Based Solid-state Drives. In *Proc. IEEE*, 2017.

[130] Jisung Park, Youngdon Jung, Jonghoon Won, Minji Kang, Sungjin Lee, and Jihong Kim. RansomeBlocker: a Low-Overhead Ransomware-Proof SSD. In *DAC*, 2019.

[131] Jisung Park, Jaeyong Jeong, Sungjin Lee, Youngsun Song, and Jihong Kim. Improving Performance and Lifetime of NAND Storage Systems Using Relaxed Program Sequence. In *DAC*, 2016.

[132] Li-Pin Chang. On Efficient Wear Leveling for Large-scale Flash-memory Storage Systems. In *ACM SAC*, 2007.

[133] Serial ATA International Organization. SATA revision 3.0 specifications. https://www.sata-io.org.

[134] Samsung. Samsung SSD 980 PRO. https://www.samsung.com/semiconductor/minisite/ssd/product/consumer/980pro/, 2020.

[135] PCI-SIG. PCI Express Base Specification Revision 3.0. https://pcisig.com/specifications.

[136] PCI-SIG. PCI Express Base Specification Revision 4.0, Version 1.0. https://pcisig.com/specifications.

[137] AMD. AMD® EPYC® 7742 CPU. https://www.amd.com/en/products/cpu/amd-epyc-7742.

[138] Micron. Micron 9300 SSD. https://www.micron.com/products/ssd/product-lines/9300, 2019.

[139] Western Digital. WD Blue SATA Internal SSD Hard Drive. https://www.westerndigital.com/en-ca/products/internal-drives/wd-blue-sata-2-5-ssd#WDS400T2B0A.

[140] Ann Franchesca Laguna, Hasindu Gamaarachchi, Xunzhao Yin, Michael Niemier, Sri Parameswaran, and X Sharon Hu. Seed-and-vote Based In-memory Accelerator for DNA Read Mapping. In *ICCAD*, 2020.

[141] Roman Kaplan, Leonid Yavits, and Ran Ginosar. RASSA: Resistive Prealignment Accelerator for Approximate DNA Long Read Mapping. *IEEE Micro*, 2018.

[142] Gunjae Koo, Kiran Kumar Matam, I Te, HV Krishna Giri Narra, Jing Li, et al. Summarizer: Trading Communication with Computing Near Storage. In *MICRO*, 2017.

[143] Vikram Sharma Mailthody, Zaid Qureshi, Weixin Liang, Ziyan Feng, Simon Garcia De Gonzalo, et al. Deepstore: In-storage Acceleration for Intelligent Queries. In *MICRO*, 2019.

[144] Valerie A Schneider, Tina Graves-Lindsay, Kerstin Howe, Nathan Bouk, Hsiu-Chuan Chen, et al. Evaluation of GRCh38 and De Novo Haploid Genome Assemblies Demonstrates the Enduring Quality of the Reference Assembly. *Genome Research*, 2017.

[145] Quynh Dang. Secure Hash Standard. https://doi.org/10.6028/NIST.FIPS.180-4, 2015.

[146] Ronald Rivest. RFC1321: The MD5 Message-digest Algorithm. https://datatracker.ietf.org/doc/rfc1321/, 1992.

[147] Nika Mansouri Ghiasi, Jisung Park, Harun Mustafa, Jeremie Kim, Ataberk Olgun, et al. GenStore: A High-Performance and Energy-Efficient In-Storage Computing System for Genome Sequence Analysis. In *arXiv*, 2022.

[148] David Sims, Ian Sudbery, Nicholas E Ilott, Andreas Heger, and Chris P Ponting. Sequencing Depth and Coverage: Key Considerations in Genomic Analyses. *Nature Reviews Genetics*, 2014.

[149] Illumina. NovaSeq 6000 System Specifications. https://emea.illumina.com/systems/sequencing-platforms/novaseq/specifications.html, 2020.

[150] Lenovo. ThinkPad T470p. https://www.lenovo.com/ch/en/laptops/thinkpad/t-series/ThinkPad-T470p/p/22TP2TT470P, 2016.

[151] Wooseong Cheong, Chanho Yoon, Seonghoon Woo, Kyuwook Han, Daehyun Kim, et al. A Flash Memory Controller for 15μs Ultra-Low-Latency SSD Using High-Speed 3D NAND Flash with 3μs Read Time. In *ISSCC*, 2018.

[152] Jisung Park, Myungsuk Kim, Myoungjun Chun, Lois Orosa, Jihong Kim, and Onur Mutlu. Reducing Solid-State Drive Read Latency by Optimizing Read-Retry. In *ASPLOS*, 2021.

[153] Florian P Breitwieser, Mihaela Pertea, Aleksey V Zimin, and Steven L Salzberg. Human Contamination in Bacterial Genomes has Created Thousands of Spurious Proteins. *Genome Research*, 2019.

[154] Human Microbiome Project Consortium et al. Structure, Function and Diversity of the Healthy Human Microbiome. *Nature*, 2012.

[155] David Danko, Daniela Bezdan, Evan E Afshin, Sofia Ahsanuddin, Chandrima Bhattacharya, et al. A Global Metagenomic Map of Urban Microbiomes and Antimicrobial Resistance. *Cell*, 2021.

[156] Rob Knight, Alison Vrbanac, Bryn C Taylor, Alexander Aksenov, Chris Callewaert, et al. Best Practices for Analysing Microbiomes. *Nature Reviews Microbiology*, 2018.

[157] Eric W Sayers, Jeffrey Beck, Evan E Bolton, Devon Bourexis, James R Brister, et al. Database Resources of the National Center for Biotechnology Information. *Nucleic Acids Research*, 2021.

[158] Justin M Zook, David Catoe, Jennifer McDaniel, Lindsay Vang, Noah Spies, et al. Extensive Sequencing of Seven Human Genomes to Characterize Benchmark Reference Materials. *Scientific data*, 2016.

[159] Karen Clark, Ilene Karsch-Mizrachi, David J Lipman, James Ostell, and Eric W Sayers. GenBank. *Nucleic Acids Research*, 2016.

[160] Fan Wu, Su Zhao, Bin Yu, Yan-Mei Chen, Wen Wang, et al. A New Coronavirus Associated with Human Respiratory Disease in China. *Nature*, 2020.

[161] Heike Sichtig, Timothy Minogue, Yi Yan, Christopher Stefan, Adrienne Hall, et al. FDA-ARGOS is A Database with Public Quality-controlled Reference Genomes for Diagnostic Use and Regulatory Science. *Nature Communications*, 2019.

[162] Stacia R Engel, Fred S Dietrich, Dianna G Fisk, Gail Binkley, Rama Balakrishnan, et al. The Reference Genome Sequence of Saccharomyces Cerevisiae: Then and Now. *G3: Genes, Genomes, Genetics*, 2014.

[163] Tanya Z Berardini, Leonore Reiser, Donghui Li, Yarik Mezheritsky, Robert Muller, et al. The Arabidopsis Information Resource: Making and Mining the "Gold Standard" Annotated Reference Plant Genome. *Genesis*, 2015.

[164] Aoife Larkin, Steven J Marygold, Giulia Antonazzo, Helen Attrill, Gilberto dos Santos, et al. FlyBase: Updates to the Drosophila Melanogaster Knowledge Base. *Nucleic Acids Research*, 2020.

[165] Deanna M Church, Leo Goodstadt, LaDeana W Hillier, Michael C Zody, Steve Goldstein, et al. Lineage-specific Biology Revealed by a Finished Genome Assembly of the Mouse. *PLoS Biology*, 2009.

[166] Thomas Wang. Integer Hash Function. http://web.archive.org/web/20071223173210/http://www.concentric.net/~Ttwang/tech/inthash.htm, 2007.

[167] Alexander Dobin, Carrie A. Davis, Felix Schlesinger, Jorg Drenkow, Chris Zaleski, et al. STAR: Ultrafast Universal RNA-seq Aligner. *Bioinformatics*, 2012.

[168] Yu Cai, Saugata Ghose, Erich F. Haratsch, Yixin Luo, and Onur Mutlu. Reliability Issues in Flash-memory-based Solid-state Drives: Experimental Analysis, Mitigation, Recovery. In *Inside Solid State Drives*. Springer, 2018.

[169] Yu Cai, Saugata Ghose, Yixin Luo, Ken Mai, Onur Mutlu, and Erich F. Haratsch. Vulnerabilities in MLC NAND Flash Memory Programming: Experimental Analysis, Exploits, and Mitigation Techniques. In *HPCA*, 2017.

[170] Yixin Luo, Saugata Ghose, Yu Cai, Erich F Haratsch, and Onur Mutlu. Improving 3D NAND Flash Memory Lifetime by Tolerating Early Retention Loss and Process Variation. *ACM POMACS*, 2018.

[171] Yixin Luo, Saugata Ghose, Yu Cai, Erich F. Haratsch, and Onur Mutlu. HeatWatch: Improving 3D NAND Flash Memory Device Reliability by Exploiting Self-recovery and Temperature Awareness. In *HPCA*, 2018.

[172] Yu Cai, Yixin Luo, Saugata Ghose, and Onur Mutlu. Read Disturb Errors in MLC NAND Flash Memory: Characterization, Mitigation, and Recovery. In *IEEE/IFIP DSN*, 2015.

[173] Yu Cai, Gulay Yalcin, Onur Mutlu, Erich F Haratsch, Adrian Crista, et al. Error Analysis and Management for MLC NAND Flash Memory. *Intel Technology*, 2013.

[174] Yu Cai, Gulay Yalcin, Onur Mutlu, Erich F Haratsch, Adrian Cristal, et al. Flash Correct-and-refresh: Retention-aware Error Management for Increased Flash Memory lifetime. In *ICCD*, 2012.

[175] Keonsoo Ha, Jaeyong Jeong, and Jihong Kim. An Integrated Approach for Managing Read Disturbs in High-density NAND Flash Memory. *IEEE TCAD*, 2015.

[176] Yixin Luo, Yu Cai, Saugata Ghose, Jongmoo Choi, and Onur Mutlu. WARM: Improving NAND Flash Memory Lifetime with Write-Hotness Aware Retention Management. In *MSST*, 2015.

[177] Micron. Product Flyer: Micron 3D NAND Flash Memory. https://www.micron.com/-/media/client/global/documents/products/product-flyer/3d_nand_flyer.pdf?la=en, 2016.

[178] Synopsys, Inc. Design Compiler. https://www.synopsys.com/implementation-and-signoff/rtl-synthesis-test/design-compiler-graphical.html.

[179] Micron Technology Inc. 4Gb: x4, x8, x16 DDR4 SDRAM Data Sheet, 2016.

[180] Saugata Ghose, Tianshi Li, Nastaran Hajinazar, Damla Senol Cali, and Onur Mutlu. Demystifying Complex Workload-DRAM Interactions: An Experimental Study. *ACM POMACS*, 2019.

[181] Saugata Ghose, Abdullah Giray Yaglikçi, Raghav Gupta, Donghyuk Lee, Kais Kudrolli, et al. What Your DRAM Power Models Are Not Telling You: Lessons from a Detailed Experimental Study. *POMACS*, 2018.

[182] Ramulator Source Code. https://github.com/CMU-SAFARI/ramulator.

[183] Advanced Micro Devices. AMD® μProf. https://developer.amd.com/amd-uprof/, 2021.

[184] M. Holtgrewe. Mason - A Read Simulator for Second Generation Sequencing Data. *Technical Report FU Berlin*, 2010.

[185] WikiChip. Cascade Lake SP - Intel. https://en.wikichip.org/wiki/intel/cores/cascade_lake_sp.

[186] Aaron Stillmaker and Bevan Baas. Scaling Equations for the Accurate Prediction of CMOS Device Performance from 180 Nm to 7 Nm. *Integration*, 2017.

[187] ARM Holdings. Cortex-R4. https://developer.arm.com/ip-products/processors/cortex-r/cortex-r4, 2011.

[188] Yongchao Liu, Douglas L Maskell, and Bertil Schmidt. CUDASW++: Optimizing Smith-Waterman Sequence Database Searches for CUDA-enabled Graphics Processing Units. *BMC Research Notes*, 2009.

[189] Yongchao Liu, Bertil Schmidt, and Douglas L Maskell. CUDASW++ 2.0: Enhanced Smith-Waterman Protein Database Search on CUDA-enabled GPUs Based on SIMT and Virtualized SIMD Abstractions. *BMC Research Notes*, 2010.

[190] Shuyi Pei, Jing Yang, and Qing Yang. REGISTOR: A Platform for Unstructured Data Processing inside SSD Storage. *ACM TOS*, 2019.

[191] Sang-Woo Jun, Andy Wright, Sizhuo Zhang, Shuotao Xu, et al. GraFBoost: Using Accelerated Flash Storage for External Graph Analytics. In *ISCA*, 2018.

[192] Jaeyoung Do, Yang-Suk Kee, Jignesh M Patel, Chanik Park, Kwanghyun Park, and David J DeWitt. Query Processing on Smart SSDs: Opportunities and Challenges. In *ACM SIGMOD*, 2013.

[193] Sudharsan Seshadri, Mark Gahagan, Sundaram Bhaskaran, Trevor Bunker, Arup De, et al. Willow: A User-Programmable SSD. In *USENIX OSDI*, 2014.

[194] Sungchan Kim, Hyunok Oh, Chanik Park, Sangyeun Cho, Sang-Won Lee, and Bongki Moon. In-storage Processing of Database Scans and Joins. *Information Sciences*, 2016.

[195] Erik Riedel, Christos Faloutsos, Garth A Gibson, and David Nagle. Active Disks for Large-Scale Data Processing. *Computer*, 2001.

[196] Erik Riedel, Garth Gibson, and Christos Faloutsos. Active Storage for Large-Scale Data Mining and Multimedia Applications. *VLDB*, 1998.

[197] Boncheol Gu, Andre S Yoon, Duck-Ho Bae, Insoon Jo, Jinyoung Lee, et al. Biscuit: A Framework for Near-data Processing of Big Data Workloads. *ISCA*, 2016.

[198] Yangwook Kang, Yang-suk Kee, Ethan L Miller, and Chanik Park. Enabling Cost-effective Data Processing with Smart SSD. In *MSST*, 2013.

[199] Xiaohao Wang, Yifan Yuan, You Zhou, Chance C Coats, and Jian Huang. Project Almanac: A Time-traveling Solid-state Drive. In *EuroSys*, 2019.

[200] Anurag Acharya, Mustafa Uysal, and Joel Saltz. Active Disks: Programming Model, Algorithms and Evaluation. *ASPLOS*, 1998.

[201] Kimberly Keeton, David A Patterson, and Joseph M Hellerstein. A Case for Intelligent Disks (IDISKs). *SIGMOD Record*, 1998.

[202] Sang-Woo Jun, Ming Liu, Sungjin Lee, Jamey Hicks, John Ankcorn, et al. Bluedbm: An Appliance for Big Data Analytics. In *ISCA*, 2015.

[203] Sang-Woo Jun, Ming Liu, Sungjin Lee, Jamey Hicks, John Ankcorn, et al. Bluedbm: Distributed Flash Storage for Big Data Analytics. *ACM TOCS*, 2016.

[204] Mahdi Torabzadehkashi, Siavash Rezaei, Ali Heydarigorji, Hosein Bobarshad, Vladimir Alves, and Nader Bagherzadeh. Catalina: In-storage Processing Acceleration for Scalable Big Data Analytics. In *Euromicro PDP*, 2019.

[205] Joo Hwan Lee, Hui Zhang, Veronica Lagrange, Praveen Krishnamoorthy, Xiaodong Zhao, and Yang Seok Ki. SmartSSD: FPGA Accelerated Near-Storage Data Analytics on SSD. *IEEE Computer Architecture Letters*, 2020.

[206] Mohammadamin Ajdari, Pyeongsu Park, Joonsung Kim, Dongup Kwon, and Jangwoo Kim. CIDR: A Cost-effective In-line Data Reduction System for Terabit-per-second Scale SSD Arrays. In *HPCA*, 2019.

[207] Benjamin Y Cho, Won Seob Jeong, Doohwan Oh, and Won Woo Ro. Xsd: Accelerating Mapreduce by Harnessing the GPU inside an SSD. In *Near-Data Processing*, 2013.

[208] Won Seob Jeong, Changmin Lee, Keunsoo Kim, Myung Kuk Yoon, Won Jeon, et al. REACT: Scalable and High-performance Regular Expression Pattern Matching Accelerator for In-storage Processing. *IEEE TPDS*, 2019.

[209] Sang-Woo Jun, Huy T Nguyen, Vijay Gadepally, et al. In-storage Embedded Accelerator for Sparse Pattern Processing. In *HPEC*, 2016.

[210] Gretchen A Morrison, Jianmin Fu, Grace C Lee, Nathan P Wiederhold, Connie F Cañete-Gibas, et al. Nanopore Sequencing of the Fungal Intergenic Spacer Sequence as a Potential Rapid Diagnostic Assay. *Journal of Clinical Microbiology*, 2020.

[211] Oxford Nanopore Technologies. R10.3: the Newest Nanopore for High Accuracy Nanopore Sequencing – Now Available in Store. https://nanoporetech.com/about-us/news/r103-newest-nanopore-high-accuracy-nanopore-sequencing-now-available-store/, 2020.

[212] Michael A Quail, Iwanka Kozarewa, Frances Smith, Aylwyn Scally, Philip J Stephens, et al. A Large Genome Center's Improvements to the Illumina Sequencing System. *Nature Methods*, 2008.

[213] PacBio Sequencing. Pacific Biosciences Closes Acquisition of Omniome and Establishes San Diego Presence. https://www.pacb.com/press_releases/pacific-biosciences-closes-acquisition-of-omniome-and-establishes-san-diego-presence/, 2021.

[214] Tim Dunn, Harisankar Sadasivan, Jack Wadden, Kush Goliya, Kuan-Yu Chen, et al. Squigglefilter: An accelerator for portable virus detection. *MICRO*, 2021.

[215] Tobias P Loka, Simon H Tausch, and Bernhard Y Renard. Reliable Variant Calling during Runtime of Illumina Sequencing. *Scientific Reports*, 2019.

[216] Haowen Zhang, Haoran Li, Chirag Jain, Haoyu Cheng, Kin Fai Au, et al. Real-time Mapping of Nanopore Raw Signals. *Bioinformatics*, 2021.

[217] Sam Kovaka, Yunfan Fan, Bohan Ni, Winston Timp, and Michael C Schatz. Targeted Nanopore Sequencing by Real-time Mapping of Raw Electrical Signal with UNCALLED. *Nature Biotechnology*, 2020.

[218] Ebrahim Afshinnekoo, Cem Meydan, Shanin Chowdhury, Dyala Jaroudi, Collin Boyer, et al. Geospatial Resolution of Human and Bacterial Diversity with City-scale Metagenomics. *Cell Systems*, 2015.

[219] Tiffany Hsu, Regina Joice, Jose Vallarino, Galeb Abu-Ali, Erica M Hartmann, et al. Urban Transit System Microbial Communities Differ by Surface Type and Interaction with Humans and the Environment. *Msystems*, 2016.

[220] Rachel M Sherman, Juliet Forman, Valentin Antonescu, Daniela Puiu, Michelle Daya, et al. Assembly of A Pan-genome from Deep Sequencing of 910 Humans of African Descent. *Nature Genetics*, 2019.

[221] Qiuhui Li, Shilin Tian, Bin Yan, Chi Man Liu, Tak-Wah Lam, et al. Building a Chinese Pan-genome of 486 Individuals. *Communications Biology*, 2021.

[222] Karen H Miga and Ting Wang. The Need for A Human Pangenome Reference Sequence. *Annual Review of Genomics and Human Genetics*, 2021.

[223] Haowen Zhang, Chirag Jain, and Srinivas Aluru. A Comprehensive Evaluation of Long Read Error Correction Methods. *BMC Genomics*, 2020.

[224] Karen H Miga, Sergey Koren, Arang Rhie, Mitchell R Vollger, Ariel Gershman, et al. Telomere-to-telomere Assembly of A Complete Human X Chromosome. *Nature*, 2020.

[225] Glennis A Logsdon, Mitchell R Vollger, PingHsun Hsieh, Yafei Mao, Mikhail A Liskovykh, et al. The Structure, Function and Evolution of A Complete Human Chromosome 8. *Nature*, 2021.

[226] Peter Perešíni, Vladimír Boža, Broňa Brejová, and Tomáš Vinař. Nanopore Base Calling on the Edge. *Bioinformatics*, 2021.

[227] Omar Ahmed, Massimiliano Rossi, Sam Kovaka, Michael C Schatz, Travis Gagie, et al. Pan-genomic Matching Statistics for Targeted Nanopore Sequencing. *iScience*, 2021.

[228] Aaron Pomerantz, Nicolás Peñafiel, Alejandro Arteaga, Lucas Bustamante, Frank Pichardo, et al. Real-time DNA Barcoding in a Rainforest Using Nanopore Sequencing: Opportunities for Rapid Biodiversity Assessments and Local Capacity Building. *GigaScience*, 2018.

[229] Shinichi Sunagawa, Luis Pedro Coelho, Samuel Chaffron, Jens Roat Kultima, Karine Labadie, et al. Structure and Function of the Global Ocean Microbiome. *Science*, 2015.

[230] Simon Lax, Daniel P Smith, Jarrad Hampton-Marcell, Sarah M Owens, Kim M Handley, et al. Longitudinal Analysis of Microbial Interaction between Humans and the Indoor Environment. *Science*, 2014.