MARS: Processing-In-Memory Acceleration of Raw Signal Genome Analysis Inside the Storage Subsystem

Melina Soysal
ETH Zurich
Zurich, Switzerland
melina.soysal@tum.de

Konstantina Koliogeorgi
ETH Zurich
Zurich, Switzerland
kkoliogeorgi@ethz.ch

Can Firtina
ETH Zurich
Zurich, Switzerland
canfirtina@gmail.com

Nika Mansouri Ghiasi
ETH Zurich
Zurich, Switzerland
n.mansorighiasi@gmail.com

Rakesh Nadig
ETH Zurich
Zurich, Switzerland
rakesh.nadig@gmail.com

Haiyu Mao
King's College London
London, United Kingdom
maohaiyu1993@gmail.com

Geraldo Francisco de Oliveira Junior
ETH Zurich
Zurich, Switzerland
geraldofojunior@gmail.com

Yu Liang
ETH Zurich
Zurich, Switzerland
yulianglenny@gmail.com

Klea Zambaku
ETH Zurich
Zurich, Switzerland
kzambaku@ethz.ch

Mohammad Sadrosadati
ETH Zürich
Zurich, Switzerland
m.sadr89@gmail.com

Onur Mutlu
ETH Zurich
Zurich, Switzerland
omutlu@gmail.com

Abstract

Conventional genome analysis relies on translating the noisy raw electrical signals generated by DNA sequencing technologies into nucleotide bases (i.e., A, C, G, and T) through a computationally-intensive process called basecalling. Raw signal genome analysis (RSGA) has emerged as a promising approach towards enabling real-time genome analysis by directly analyzing raw electrical signals without the need for basecalling. However, rapid advancements in sequencing technologies make it increasingly difficult for software-based RSGA to match the throughput of raw signal generation. Hardware-based RSGA acceleration has the potential to bridge the gap between software-based RSGA and sequencing throughput.

This paper demonstrates that while (i) conventional hardware acceleration techniques (e.g., specialized ASICs) in tandem with (ii) memory-centric approaches (e.g., Processing-In-Memory) can significantly accelerate RSGA, the high volume of genomic data greatly shifts the performance and energy bottleneck from computation to I/O data movement. As sequencing throughput increases, I/O overhead becomes the dominant contributor to both runtime and energy consumption, limiting the scalability of both processor-centric and main-memory-centric accelerators. Therefore, there is a pressing need to design a high-performance, energy-efficient system for RSGA that can both alleviate the data movement bottleneck and provide large acceleration capabilities.

We propose MARS, a storage-centric system that leverages the heterogeneous resources available within modern storage systems (e.g., storage-internal DRAM, storage controller, flash chips) alongside their large storage capacity to tackle *both* data movement and computational overheads of RSGA in an area-efficient and low-cost manner. MARS accelerates RSGA through a novel hardware/software co-design approach using three major techniques. First, MARS modifies the RSGA pipeline via a previously unexplored combination of two filtering mechanisms and a quantization scheme, reducing hardware demands and optimizing for in-storage execution. Second, MARS accelerates the modified

ICS '25, Salt Lake City, UT, USA

ACM ISBN 979-8-4007-1537-2/25/06
https://doi.org/10.1145/3721145.3730428

RSGA steps directly within the storage device by leveraging both Processing-Near-Memory and Processing-Using-Memory paradigms, tailored to the internal architecture of the storage system. Third, MARS orchestrates the execution of all steps via a streamlined control and data flow to fully exploit in-storage parallelism and minimize data movement. Our evaluation shows that MARS outperforms basecalling-based software and hardware-accelerated state-of-the-art read mapping pipelines by 93× and 40×, on average across different datasets, while reducing their energy consumption by 427× and 72×. MARS improves the performance of state-of-the-art RSGA-based read mapping pipeline by 28× while reducing its energy consumption by 180× on average across different datasets.

CCS Concepts

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

Keywords

Processing-In-Memory, Genome Analysis, In Storage Processing, Processing-Near-Memory

ACM Reference Format:

Melina Soysal, Konstantina Koliogeorgi, Can Firtina, Nika Mansouri Ghiasi, Rakesh Nadig, Haiyu Mao, Geraldo Francisco de Oliveira Junior, Yu Liang, Klea Zambaku, Mohammad Sadrosadati, and Onur Mutlu. 2025. MARS: Processing-In-Memory Acceleration of Raw Signal Genome Analysis Inside the Storage Subsystem. In *2025 International Conference on Supercomputing (ICS '25), June 08–11, 2025, Salt Lake City, UT, USA.* ACM, New York, NY, USA, 22 pages. https://doi.org/10.1145/3721145.3730428

1 Introduction

Identifying and analyzing an organism's DNA sequence, i.e., *genome analysis*, has led to important advances in areas such as personalized medicine [33, 71, 201], outbreak tracing [39, 229], and evolutionary biology [62, 169, 175]. *Genome sequencing* is the experimental process of determining the nucleotide sequence of an organism's DNA. As current technologies cannot generate a single long sequence for an entire genome, DNA is first fragmented into short sequences, called *reads*, which serve as input to computational analyses [1, 20, 36, 68, 119, 128, 129, 135, 168, 234] to reconstruct the genome and extract biological insights. The analysis typically starts with *mapping* reads to a known *reference genome* [21, 26], followed by identifying mutations and other genetic variations [1, 36, 38, 68, 119, 128, 129, 135, 168, 234] during downstream analyses.

Nanopore sequencing technology [53, 103, 104, 122, 123, 186, 234] enables DNA sequencing by passing DNA strands through nano-scale pores, known as *nanopores*, and measuring the resulting fluctuations in electrical current. These current fluctuations, referred to as *raw signals*, correspond to distinct sequences of DNA nucleotides and form the basis for downstream analyses. The small dimensions of the nanopores enable sequencing in compact devices [103], paving the way for portable, scalable, and low-cost [189] sequencing for a wide range of applications, including outbreak tracing and disease diagnosis [77, 108]. Nanopore sequencers' rapid adoption is further driven by their unique capability of early termination of sequencing when further data is no longer needed [144, 164], **reducing the sequencing time and cost** and enabling **real-time** analysis [68, 190].

Typical genome analysis pipelines first translate noisy raw electrical signals into sequences of nucleobase characters through a process called basecalling [1, 96, 186, 196]. Subsequent downstream analyses are then performed on these text-based sequences. However, basecalling is computationally intensive and represents a major bottleneck for real-time analysis, as it relies heavily on sophisticated deep learning models [60, 153, 196, 234]. Given the increasing demand for real-time processing, there is a pressing need for developing fundamentally new algorithmic approaches to keep up with the rapid advances in nanopore sequencing in terms of performance, energy consumption, and cost [60, 68, 70, 135, 215].

Raw signal genome analysis (RSGA) [36, 60, 61, 68–70, 119, 135, 159, 164, 179, 190, 193, 234] has been proposed as a new paradigm that bypasses traditional basecalling by operating *directly* on raw electrical signals. Instead of translating signals into nucleotide sequences, RSGA analyzes the raw signals themselves to perform genomic tasks such as read mapping and variant detection. RSGA can complement basecalling by serving as a lightweight pre-basecalling filter [45] to reduce redundant basecalling operations or even replace basecalling entirely by directly analyzing raw signals **in real-time** without translating them to nucleotide sequences first [60, 68, 70, 135]. RSGA can lead to more comprehensive [119, 135, 220] genome analysis as it preserves richer sequencing information in the raw signals [19, 135, 170, 194, 198, 210, 217]. These key benefits have fueled rapid research progress in the field of RSGA [36, 68, 70, 119, 135, 159, 164, 179, 193, 234], opening new directions such as direct alignment [60, 120, 135] and de novo assembly [69] on raw signals.

As advancements in sequencing technologies continue at a rapid pace, scalability challenges arise, placing increasing pressure on software-based RSGA to match the throughput of raw signal generation and meet the real-time requirements. To bridge the widening gap between sequencing throughput and downstream analysis, hardware acceleration is required to either process larger data volumes

with the same computational resources or reduce execution time and energy consumption. Research efforts have targeted the computational bottlenecks of RSGA by using GPUs (e.g., [36, 74, 82, 178, 184]) or co-designing algorithms with specialized hardware architectures and ASICs (e.g., [60, 141, 180, 190, 200]). While these approaches effectively reduce computational overhead, they largely overlook the impact of I/O data movement from the storage subsystem on the *end-to-end* RSGA pipeline. Our motivational analysis (§3) shows that as computational bottlenecks of RSGA are accelerated, the contribution of I/O becomes dominant and ultimately emerges as the primary bottleneck in the end-to-end analysis. For instance, as the accelerator speedups increase, the adverse impact of the storage subsystem dominates the accelerated end-to-end execution latency, reaching up to 78% of total execution time for large genomes (see §3.2). This motivational study highlights the need for an architecture for RSGA that (i) alleviates the large data movement overhead, (ii) accelerates the computational steps of RSGA, and (iii) scales to the large volumes of genomic datasets.

Our goal in this work is to design a high-performance, energy-efficient, and scalable system for RSGA by effectively addressing *both* the data movement and computation overheads of the end-to-end RSGA pipeline for read mapping. Our key idea is to design a *storage-centric* system that leverages the *heterogeneous compute-capable resources* (e.g., SSD internal DRAM, SSD controller), alongside the large storage capacity available within modern storage systems to alleviate I/O data movement and computational bottlenecks within the RSGA pipeline in an area-efficient and low-cost manner. To this end, we propose *MARS* (Processing-In-**M**emory **A**cceleration of **R**aw Signal Genome Analysis Inside the **S**torage Subsystem), the first In-Storage-Processing (ISP) design combining Processing-Using-DRAM and Processing-Near-DRAM ***within*** a storage system.

Challenges. Despite ISP's promising potential, designing a storage-centric system for RSGA presents several key challenges. First, RSGA steps (e.g., event detection, seeding and chaining) exhibit high memory demands and irregular data access patterns. In contrast, SSDs lack architectural support for fine-grained (i.e., small-size) memory operations and are optimized for sequential access to fully utilize the high flash memory channel bandwidth inside the SSD. Second, exploiting heterogeneous resources and computation capabilities within the storage system introduces a complex design space and a rich set of tuning parameters. Third, deploying the end-to-end RSGA pipeline consisting of multiple steps inside the storage system creates contention over shared resources, requiring careful coordination and isolation. Addressing these challenges necessitates a carefully-constructed design to ensure a synergistic and efficient orchestration of the available in-storage resources.

We address these challenges through a novel hardware/-software co-design approach that modifies and enables RSGA computational primitives to leverage in-storage execution capabilities, while carefully taking into account storage system constraints. First, we propose two software modifications: (1) a novel combination of two filtering mechanisms [70, 128, 133, 136] that selectively remove redundant or low-quality candidate matches between the input and reference genomes early in the RSGA pipeline, reducing both computational workload and intermediate data storage requirements and (2) an arithmetic conversion scheme that reduces the precision of intermediate signal representations to lower storage and computation overheads, carefully placed in the RSGA pipeline to preserve accuracy. Second, we augment the storage system's functionality to support the RSGA pipeline by placing accelerators for individual steps in different parts of the storage subsystem, leveraging different *'Processing-In-Memory'* paradigms: (i) inside the memory array of the storage-internal DRAM through the *'Processing-Using-DRAM'* approach, (ii) near the subarrays of the storage-internal DRAM using the *'Processing-Near-DRAM'* approach and (iii) inside the storage controller via the *'Processing-Near-DRAM'* approach, which operates on data fetched from the storage-internal DRAM. MARS orchestrates these individual components through a unified control and data flow that minimizes data movement and efficiently exploits the available bandwidth between them.

We evaluate MARS-based read mapping in terms of accuracy, latency and energy consumption across five diverse genomic input datasets from different species. We compare our design against four state-of-the-art software and hardware baselines using both RSGA and basecalling-based approaches and make four major observations. First, MARS outperforms the state-of-the-art CPU-based RSGA implementation for read mapping [70] by 28× on average across all datasets while improving the energy consumption by 180× on average. Second, MARS provides an average speedup of 93× over a hybrid CPU/GPU-accelerated basecalling-based pipeline [2, 128], while improving energy consumption by 427× on average. Third, MARS is superior to GenPIP [153], a state-of-the-art Processing-In-Memory-based read mapping system relying on basecalling, achieving a speedup of 40× and energy savings of 72× on average across all five datasets. Fourth, MARS provides analysis accuracy *on par* with the conventional basecalling-based pipeline.

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

- It is the first work to demonstrate the I/O bottleneck of hardware-accelerated Raw Signal Genome Analysis (RSGA) and propose In-Storage-Processing of RSGA.
- We propose MARS, the *first* In-Storage-Processing system for RSGA, which mitigates both I/O data movement

and computational overheads through a tightly integrated hardware/software co-design.

- To our knowledge, MARS is the first architecture to integrate *multiple Processing-In-Memory paradigms* within the storage system. We implement accelerators *inside* the SSD's DRAM, *near the subarrays* of the SSD's DRAM as well as *inside the SSD controller* leveraging both Processing-Using-DRAM and Processing-Near-DRAM paradigms to efficiently enable diverse RSGA computation primitives.
- We extensively compare MARS to state-of-the-art software and hardware baselines that use both RSGA and basecalling. We show that MARS improves performance over software and hardware-accelerated state-of-the-art read mapping pipelines by a factor of 93× and 40× while reducing their energy consumption by 427× and 72× on average across five real-world datasets.

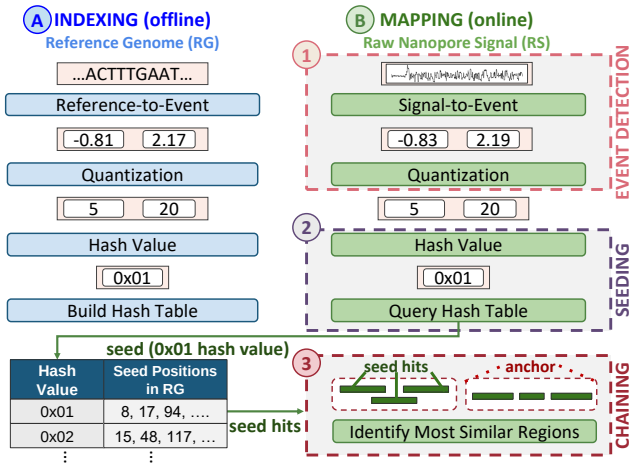

Figure 1: Overview of a typical RSGA read mapping workflow based on a hash-table for indexing.

2 Background

2.1 Genome Analysis

Raw Signal Genome Analysis. Nanopore sequencing can sequence relatively long fragments of DNA [53, 103, 104, 122, 123, 186, 234], called *reads*, by measuring the electrical current changes caused when a DNA fragment traverses a tiny pore, called *nanopore*. The generated sequence data [53, 103, 104, 122, 123, 186, 234], referred to as *raw signals*, are then used in downstream analysis, e.g., for read mapping [36, 68, 70, 119, 190, 234] and alignment [60, 120, 135] purposes. In the conventional genome analysis approach, raw signals are first translated into sequences of nucleobase characters (i.e., A, C, G, T) during the basecalling process [1, 2, 24, 26, 45], and then mapped to a reference genome to find similarities and differences [1, 96, 195, 196, 228, 230]. In contrast, RSGA eliminates the need for basecalling by directly operating on raw signals [36, 60, 68, 70, 119, 159, 179, 193, 234].

RSGA requires comparing sequences from the reference genome with sequences derived from each input query, i.e., the raw electrical signals generated by the sequencer for a given DNA sample. To enable this comparison, both reference subsequences and raw signals are converted into *events*, i.e., a series of values corresponding to genomic subsequences of certain length. These event sequences are then passed through a quantization step that accounts for sequencing noise and enables robust signal-domain comparisons between reference and input query. A typical state-of-the-art RSGA pipeline for read mapping, illustrated in Fig. 1, consists of two main stages Ⓐ Indexing and Ⓑ Mapping.

Ⓐ Indexing (offline): The reference genome is converted into events through reference-to-event conversion and quantization. These events are then stored in an efficient data structure, e.g., a hash table, to enable fast lookup of matching signal patterns. Ⓑ Mapping (online): This stage maps raw signals to the reference genome using the previously constructed index. The first step ① in mapping is *event detection*, which performs the signal-to-event conversion of raw signals and applies quantization. In the second step ②, called *seeding*, consecutive events are grouped to generate hash values which represent signal segments known as *seeds* and are used to query the reference index. Matching entries, referred to as *seed hits*, represent candidate matches between the input and reference. During the last step ③, called *chaining*, seeds are sorted based on their positions in the reference genome. Seeds that are both spatially close and colinear, i.e., those that maintain consistent relative positions in the reference and input query, are grouped into *anchors*, which form the basis for constructing *chains* representing high-confidence matching regions between the query and the reference genome.

Filtering Techniques. Filtering techniques [22, 23, 25, 27, 38, 86, 115, 121, 128, 133, 161, 174, 225–227] are extensively used in genome analysis pipelines to reduce the need for costly alignment operations by eliminating unlikely candidate matches early during the read mapping process. One popular filtering approach, adopted in both conventional [128] and RSGA [70] approaches, is *frequency filtering*. The goal of frequency filtering is to identify and eliminate the seeds that cause a large number of seed hits in the reference genome. These frequent seed hits usually appear due to repetitions in the genome or hash collisions, which can cause ambiguity [67] in read mapping and increase the computational cost of the subsequent steps [26], such as chaining. To eliminate these issues, these seed hits are *not* considered in the subsequent chaining stage, effectively reducing the computational load. A dataset-specific value defines the threshold for filtering out such frequent matches. Another promising method is the *seed-and-vote*

filtering technique [133, 136] that has been applied in conventional basecalling-based pipelines to discard anchors that are unlikely to generate valid alignments. As shown in Fig. 2, the reference genome is partitioned into overlapping, equal-length windows W_i. Each anchor votes for the window(s) it appears in (see orange $X's$ in Fig. 2). We define as *voting threshold* the minimum number of votes per window, so that it contains correct alignments. A region whose vote count falls below this predefined threshold is excluded from further analysis, reducing the computational load of chaining.

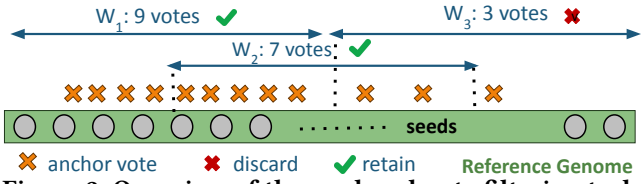

Figure 2: Overview of the seed-and-vote filtering technique for a threshold value of 5.

2.2 SSD Architecture

Fig. 3 depicts the architecture of a typical modern NAND flash-memory-based Solid State Drive (SSD) [156], which consists of three main components: (1) an array of NAND flash chips, (2) SSD controller, and (3) DRAM.

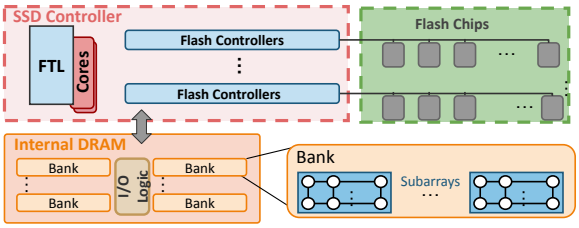

Figure 3: Organizational overview of a modern SSD.

NAND Flash Memory. NAND flash memory consists of multiple flash chips [16, 41], which are connected to the SSD controller via multiple parallel flash channels. Each flash chip typically contains one or more independent dies. Each die has multiple (e.g., 2 or 4) planes and each plane contains thousands of blocks. A block includes hundreds to thousands of pages, each of which is 4–16 KiB in size.

SSD Controller. The SSD controller [41, 42, 156] consists of two primary components: (1) multiple general-purpose cores running the SSD firmware, i.e., the *flash translation layer (FTL)*, and (2) per-channel hardware *flash controllers*. The FTL manages communication with the host system, maintains logical-to-physical (L2P) address mappings for read operations, handles internal I/O scheduling, and performs various SSD management functions to hide the complexities of NAND flash memory from the host processor. Flash controllers handle (i) requests between the SSD controller and the flash chips and (ii) error-correcting codes ECC for the NAND flash chips [41, 42, 203, 235].

SSD-Internal DRAM. Modern SSDs employ DRAM to store metadata crucial for SSD management (e.g., L2P page mapping table) and to cache frequently accessed pages [83, 134, 155, 192, 204, 237]. Typically, the DRAM takes up 0.1% of the SSD's capacity (e.g., 4GB LPDDR4 DRAM [105] for a 4TB SSD [182]). As shown in Fig. 4, DRAM is organized in a hierarchical structure. At the highest level, a DRAM module comprises multiple chips, each containing several banks (e.g., 8-16), subdivided into multiple subarrays (e.g., 64-128). A subarray is a 2D array of cells organized into multiple rows (e.g., 512-1024) and columns (e.g., 2-8kB) [116, 124]. Cells in a row share a wordline while cells in the same column share a bitline. The bitline is used to read from and write to the cells via the row buffer, which contains sense amplifiers (SA in Fig.4).

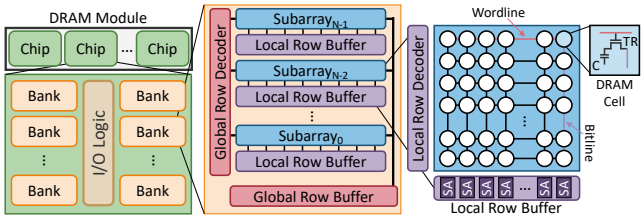

Figure 4: Organizational overview of a DRAM module.

SSD I/O Bandwidth. SSDs are characterized by the external and internal bandwidth (BW). The external BW, e.g., PCIe [63, 165] lane BW, refers to the data transfer rate between the SSD and the host system and is determined by the number of PCIe lanes. In contrast, the internal BW refers to the bandwidth between the NAND flash chips and the SSD controller. The internal BW typically exceeds the external BW. For example, recent enterprise SSD controllers [29] support 6.55GB/s external and 19.2GB/s internal BW, distributed over 16 channels operating at 1.2 GB/s each [109]. To bridge the performance gap between main memory and storage systems, modern SSDs integrate cutting-edge PCIe-Gen4 interfaces, e.g., 7 GB/s PCIe in Samsung PM1735 [183].

3 Motivation and Key idea

3.1 Computational Requirements of RSGA

RSGA is a promising approach for bridging the performance gap between sequencing technologies, such as Nanopore sequencing [53, 103, 104, 122, 123, 186, 234], and analysis times. It can reduce the basecalling workload by serving as a pre-basecalling filtering approach [45] or enable real-time analysis by completely bypassing the costly deep-learning based basecalling step [1, 2, 96, 195, 196, 228, 230]. However, given the rapid growth of sequencing throughput, it becomes exceedingly challenging for software-based RSGA to meet the requirements of real-time analysis [60]. The increasing number of flow cells and nanopores per flow cell lead to scalability challenges in processing generated data

simultaneously and in real-time. Real-time RSGA [36, 60, 68, 70, 144, 190, 234], particularly for large genomes and extensive data sets, requires medium to large-sized server-grade systems to meet the significant computational and memory needs [68, 119, 234]. For example, mapping a human genome with RSGA on our server-grade system (configuration in §7) requires 52 CPU threads and 128 GB DRAM capacity to meet the real-time analysis requirements of a single portable palm-sized sequencing device [70]. Recent state-of-the-art works [60, 190] meet real-time requirements for small genomes, but fail to scale to larger inputs due to the computationally costly operations of full-genome *alignment*, which slows down the system at quadratic rates as the genome size increases. To further understand the acceleration obstacles of RSGA workflows and exploit the full potential for acceleration, a systematic analysis is required.

We focus on RawHash2 [68], the state-of-the-art RSGA pipeline for read mapping that uses efficient quantization and a lightweight hash-based similarity search to scale to larger genomes. We choose RawHash2 as it introduces a highly efficient seed search mechanism, that leads to a better accuracy-throughput trade-off in comparison to prior RSGA read mapping mechanisms, Sigmap [234], UNCALLED [119] and RawHash [68]. We execute RawHash2 on a high-end, latency-optimized SSD [183] with a PCIe Gen4 interface (`PCIe`) [165]. Fig. 5 shows the breakdown of RawHash2 into the steps described in §2 (i.e., event detection, seeding, chaining) as well as I/O overhead. We measure I/O overhead by executing the pipeline once with data fully preloaded in memory (i.e., without I/O overhead), and once with no data preloaded into memory (i.e., with full I/O overhead from storage). The difference in total runtime between the two runs reflects the I/O data movement time from SSD to memory. We use five different datasets as inputs, enumerated from the smallest (D1, viral SARS-CoV-2 genome) to the largest one (D5, human genome). For *all* genome sizes, chaining is consistently a primary computational overhead, contributing between 33.1% (D1) and 94.9% (D5) of the total execution time. Seeding takes up 4.3%-9.3% of the execution time. Event detection and I/O data overhead are considerable bottlenecks especially for small datasets (D1, D2, D3), taking up to 20.48% and 40.84% of the execution time respectively.

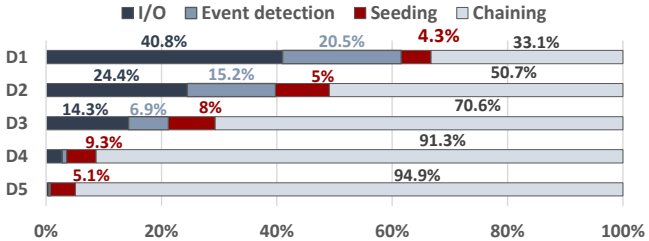

Figure 5: RawHash2 runtime breakdown for real-world genomic datasets, from smallest (D1) to largest (D5).

While no prior work has accelerated the full RSGA read mapping pipeline end-to-end, several of its individual compute primitives have been the focus of hardware acceleration efforts [21, 24, 66, 160]. **Chaining acceleration** has received significant attention in the literature. Researchers have employed GPUs [82, 177] as well as FPGAs and custom hardware architectures [80, 82, 141, 143, 190], achieving performance improvements ranging from 5.4× to 277× compared to their respective software baselines. More recently, novel computing paradigms have been explored to accelerate chaining, including PIM architectures [46] and RISC-V custom instructions [142]. **Seeding acceleration** has similarly been investigated. Custom hardware designs [93] and GPU-based implementations [52, 132] have demonstrated the potential for significant performance gains. In particular, hash-based seeding has emerged as a promising target for in-memory acceleration, with several works proposing PIM-based solutions [97, 99, 100, 102, 232, 239]. For example, [153] implements a ReRAM-based accelerator for hash-based seeding within basecalling pipelines, leveraging similar compute primitives as the seeding step in RSGA. pLUTo [65], an in-DRAM accelerator optimized for lookup-table (LUT) operations, is a promising approach for accelerating hash-based seeding and achieves speedups of up to 700× over CPU baselines for seeding-relevant workloads.

3.2 Impact of Data Movement on Hardware Accelerated RSGA

Despite the promising results of these standalone accelerators, there are no mature end-to-end accelerated systems for RSGA. Current works overlook the impact of storage I/O on the end-to-end accelerated system. As more RSGA pipeline steps are accelerated to meet the real-time requirements and the growing throughput of modern sequencing devices, time distribution across RSGA read mapping will change drastically. We expect I/O to emerge as the dominant bottleneck in the end-to-end analysis as the computational steps are increasingly accelerated and thus minimized.

We validate this hypothesis through a motivational experiment that analyses RawHash2 [68] using the same real-world datasets and hardware setup as introduced in our previous experiment (§3.1). We assume a scenario that applies state-of-the-art accelerators to the two most frequently accelerated steps: seeding and chaining. We model the latency of the accelerated workflow by incrementally reducing the latency of the seeding and chaining steps by 10% until we reach 100% total latency reduction, i.e., zero execution time. The results are shown in Fig. 6.

Fig. 6 shows how I/O data movement overhead progressively dominates end-to-end execution time as latency reduction increases. We make the key observation that as latency

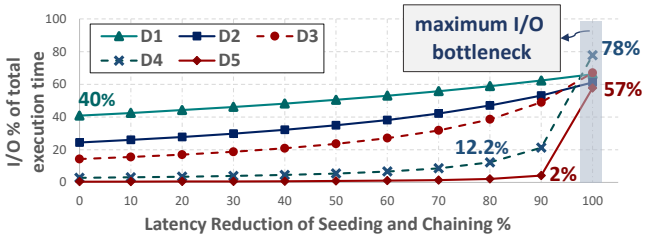

Figure 6: Impact of I/O on overall execution time under increasing acceleration of computation bottlenecks.

reduction increases, the I/O data overhead becomes the limiting factor across all datasets. In particular, for small genome datasets (D1-D3), I/O overhead reaches up to 66% of the total execution time. For larger genomes, I/O overhead remains modest until a large latency reduction of 90%. However, as computation bottlenecks are further minimized, storage I/O emerges as the primary performance limiter. **For example, I/O overhead accounts for 57% and 78% of the total execution time for D5 and D4 respectively when execution time of seeding and chaining is reduced by 100%.** These results indicate that accelerating the seeding and chaining alone is insufficient, and that I/O data movement from SSDs becomes the dominant overhead in accelerated RSGA.

3.3 Our Goal

Based on our motivational analysis, acceleration of RSGA is critical for achieving real-time genome analysis. While computational complexity is a key challenge, I/O data movement from SSDs becomes the dominant bottleneck across all genome datasets once computational steps are heavily accelerated. *In-Storage processing* (ISP) can, therefore, be a key enabler for designing a real-time system for RSGA. Specifically, ISP can uniquely address the I/O data movement bottleneck, manage the high volume of genomic data, and provide fine-grained parallelism to accelerate the computational steps. However, designing an ISP system for RSGA is challenging due to architectural constraints of SSDs. Limited hardware resources (such as main memory capacity) and inefficient random accesses prevent a straightforward implementation of the RSGA pipeline inside storage. **Our goal** in this work is to leverage ISP capabilities in a careful way to accelerate RSGA by alleviating the I/O overheads and accelerating key computational steps.

4 MARS Key Idea

The core design idea for MARS is to enable multiple Processing-In-Memory paradigms within the SSD and leverage the high SSD-internal flash channel bandwidth to create a highly-parallel heterogeneous computing environment for RSGA inside the storage system. Our optimization strategy consists of two key components: First, we propose targeted software modifications on existing RSGA pipelines that take into account SSD limitations and parallelization capabilities while maintaining accuracy. Second, we provide specialized near-memory computation units within the SSD for individual computational steps of the RSGA pipeline and orchestrate the data flow between them. We leverage two computational approaches within the SSD: (i) Processing-Using-DRAM, which exploits the analog properties of DRAM arrays in SSD to perform massively parallel in-memory operations with minimal data movement overhead. (ii) Processing-Near-DRAM, which adds lightweight compute logic *close* to the SSD's internal DRAM, either *near the DRAM subarrays* or *inside the SSD controller*, tailored to the demands of each RSGA step.

5 MARS Genome Analysis Workflow

MARS implements a genome analysis workflow based on the state-of-the-art RSGA approach presented in Fig. 1. The scope of MARS's software modifications is to reduce both computational workload and intermediate storage requirements, resulting in a version of RSGA that is optimized for efficient in-storage execution.

5.1 Filtering Techniques

We adopt two distinct filtering techniques to reduce the load on the computationally intensive and resource-demanding chaining step.

Frequency Filters. First, we leverage *frequency filters* [70, 128] to only examine unique, meaningful matches between signal queries, e.g., seeds, and reference genomes. Frequency filters are applied to the hash values created by multiple seeds (§2.1), and discard seeds that appear within the reference genome above a predefined threshold frequency ($thresh\_freq.$).

Seed-and-Vote Filtering Second, we adopt the *seed-and-vote* filtering technique [133, 136] to discard anchors unlikely to generate a correct alignment. As described in Section 2, we partition the reference genome into windows, and anchors vote for windows that contain exact matches. A window with a high number of votes is more likely to contain the correct alignment. Only windows receiving a number of votes above ($thresh\_voting$) are retained for further processing. This threshold is selected to balance accuracy (measured via F1-score) and performance, ensuring sufficient anchors are preserved for sensitivity, while discarding redundant matches to reduce workload. This is the first work to apply the seed-and-vote technique to raw signals. For raw signals, this process is particularly challenging because reads and references, when converted to events, can include noise. To address this, we apply the seed-and-vote technique *after the quantization and hash-table query steps*, to preserve accuracy.

Based on the size and characteristics of the target genome, parameter values for both filtering techniques, i.e., *thresh_freq*, *thresh_voting* and the window size for seed-and-vote filtering may vary. To ensure robust performance across a wide range of datasets, we perform an offline parameter space exploration to tune the parameter values and achieve a fair trade-off between accuracy and performance of the analysis. Our exploration space is defined by the tuple (*thresh_freq*, *thresh_voting*, *voting_window*). We test different configurations on a subset of each dataset (0.5-2%) and observe that genomes with similar properties (e.g., size or complexity) consistently benefit from the same parameter configurations. Small genomes yield the best trade-off between accuracy and performance across different datasets for values of $(2000, 5, 256)$ and large genomes for $(20000, 2, 256)$. Although the values cover a representative set of diverse genomes, they are easily reconfigurable for new genome types. The parameter exploration is performed only once offline and therefore does not impact the end-to-end runtime and energy.

5.2 Arithmetic Conversion Techniques

We improve the utilization of the internal flash-channel bandwidth available within the SSD by using arithmetic conversion techniques. The key idea of this optimization is to convert **floating-point** values to **fixed-point** and benefit from reduced storage requirements (i.e., mostly reduced bit-width from 64 or 32 bits to 16 bits) for intermediate data, as well as enable resource-efficient and less time-consuming fixed-point operations. We perform an experimental analysis at software level and evaluate the accuracy achieved for fixed-point arithmetic using 32, 16 and 8 bytes. The use of 16 bytes leads to small accuracy loss compared to floating point and significant resource utilization savings.

Our goal is to maximize savings by applying arithmetic conversion as early as possible in the pipeline. However, adopting fixed-point arithmetic at the beginning of the pipeline is challenging due to the noise of raw signals, i.e., leveraging fewer bits for raw signals interferes with subsequent signal-to-event conversion and quantization leveraged in typical RSGA pipelines [70], leading to much lower accuracy. Applying early quantization, i.e., applying quantization directly on the raw signal before signal-to-event conversion, alleviates this challenge. It increases stability against possible noise and facilitates the adoption of fixed-point arithmetic. Unlike previous works [70], our workflow first applies quantization, followed by converting floating-point to fixed-point arithmetic, and then executes the signal-to-event conversion. We show the accuracy results of our implementation for both fixed- and floating-point in §8.

6 MARS Architecture and System

We propose MARS, the first ISP system designed for accelerating RSGA by reducing data movement overheads and leveraging highly parallel computation capabilities present inside modern storage systems. We design MARS as an end-to-end In-Storage-Processing system that expands the capabilities of state-of-the-art SSDs and autonomously executes the RSGA pipeline without host intervention.

6.1 MARS In-Storage Architecture

6.1.1 ***Overview.*** Fig. 7 shows a high-level overview of our system and the application flow.[1] MARS consists of five types of components: *MARS Control Unit* Ⓘ, *Sorter Unit* Ⓘ, *Merger Unit* Ⓘ, *Arithmetic Unit* Ⓘ and *Querying Unit* Ⓥ.

SSD Controller Components: *MARS Control Unit*, *Sorter Unit* and *Merger Unit* are placed inside the SSD controller. *MARS Control Unit* Ⓘ acts as a Finite State Machine (FSM) that controls and coordinates the data flow between MARS's computation units. MARS's *Sorter Unit* Ⓘ (§6.4) is an accelerator that sorts sequences up to a predefined length. The *Merger Unit* Ⓘ (§6.4) efficiently combines short sorted sequences into longer ones. Both units follow the Processing-Near-DRAM approach, operating on data that originates from SSD-internal DRAM. One Sorter and Merger pair is added per Flash Controller, adding up to 8 instances.

SSD-internal DRAM Components: The *Arithmetic Unit* Ⓘ and *Querying Unit* Ⓥ are placed inside the SSD-internal DRAM chips. The *Arithmetic Unit* Ⓘ (§6.2) performs arithmetic and logical operations. It leverages the Processing-Near-DRAM approach: An Arithmetic Unit is placed at the edge of each pair of subarrays' peripheral logic, leading to 256 instances. The *Querying Unit* Ⓥ (§6.3) performs efficient hash-table lookups. It leverages the Processing-Using-DRAM paradigm: It exploits the analog operational properties of the SSD-internal DRAM. We implement one *Querying Unit* per subarray, i.e., 512 instances.

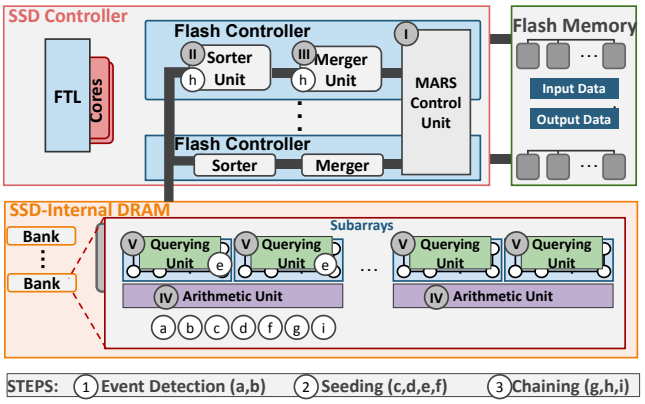

Figure 7: High-level overview of MARS architecture.

[1] To ease readability, Fig. 7 and 10 exclude control paths.

6.1.2 ***Mapping Workflow to Compute Units.*** We perform a detailed analysis of the RSGA workflow to partition the RSGA steps (i.e., event detection, seeding, chaining) into more fine-grained tasks such as arithmetic (e.g., addition, multiplication, division), querying and sorting operations. The entire RSGA workflow is described as a pipeline of these fine-grained tasks and each one is mapped to one of the available computation units (Arithmetic, Querying, Sorter, or Merger Unit) for efficient execution. The MARS Control Unit encodes the pipeline steps and the order of their execution into a Finite State Machine and sequentially orchestrates them at runtime. While the pipeline sequence is predefined, actual computations in each step are triggered dynamically based on the availability of inputs. Each compute unit is activated only when its inputs are available, ensuring resource efficiency and avoiding contention.

6.1.3 ***Control and Data Flow.*** Each step in MARS's pipeline begins as soon as the previous one finishes. Before starting execution, the index and raw input data are distributed uniformly in terms of size across all SSD channels. Data is transferred to the MARS's *Arithmetic Units* (IV), close to the SSD-internal DRAM subarrays. The *Arithmetic Units* (IV) perform the *event detection* step ① consisting of signal-to-event conversion (1a) and quantization (1b), executed sequentially. Next, as part of the seeding step ②, the *Arithmetic Units* execute the hash-value generation (c) and the frequency filter (d). The filtered hash values are used for querying (e) the hash-table for seed hits inside the DRAM at the *Querying Units* (V). The seed-and-vote filtering (f) step discards non-promising seed hits by leveraging once more the *Arithmetic Units* (IV). During the chaining step ③, the data is first bucketized (g) within the *Arithmetic Units* (IV) and transferred to the *Sorter* (II) and *Merger Units* (III) inside the SSD controller for the sorting (h) step. The sorted data fragments are consolidated back in the SSD-internal DRAM for the final part of chaining, i.e., a dynamic programming-based algorithm (3i) implemented within the *Arithmetic Units* (IV).

6.2 Event Detection Implementation

We map event detection (i.e., signal-to-event conversion and quantization §2.1) to the *Arithmetic Unit* as it mainly comprises additions and multiplications. One *Arithmetic Unit* is placed next to two SSD-internal DRAM subarrays to perform arithmetic operations close to the data and leverage the large subarray-level parallelism available within the DRAM. MARS is the first work to implement Processing-Near-DRAM inside the storage-internal DRAM.

Arithmetic Unit Architecture and Mechanism. Our design is inspired by a previous DRAM-based design, FULCRUM [126]. Fig. 8 illustrates the main components of the design. A single-word *ALU* ① is placed next to a DRAM subarray and performs addition, comparison, multiplication, and bitwise operations. *Registers* ② are placed near the ALU to store intermediate results. A programmable *Instruction Buffer* ③ stores pre-decoded information for potential instructions, i.e., different operands and branch outcomes. *Column-Selection Latches* ④ are placed on each column of each subarray to enable sequential access to individual columns [126]. A *Control Unit* ⑤ determines the order of instructions and location of next access to the memory array.

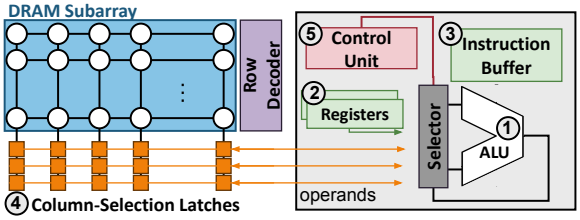

Figure 8: Overview of the MARS's Arithmetic Unit near a DRAM subarray

In order to map the operations of signal-to-event conversion and quantization to the Arithmetic Unit, we first break each of them down into arithmetic, predicate-based and condition-based operations. We construct pre-decoded instructions for all potential branches of execution within these operations and store them in the programmable instruction buffer. Based on the outcome of the previous operation, the Control Unit (i) selects the next instruction from the Instruction Buffer and (ii) identifies the columns of the subarray that need to be accessed. This ensures that the Column-Selection Latches either capture the correct input operands (read from the subarray) or hold the correct target values before writing them back to the subarray.

6.3 Seeding Implementation

The hash-value generation, frequency filter and seed-and-vote filtering steps in seeding comprise arithmetic operations and pairwise comparisons. To execute those operations efficiently, we use the *Arithmetic Unit* described in Section 6.2. However, hash table querying presents unique challenges due to the hash table's large size and the frequent random memory accesses it requires. To address this, we implement the hash-querying mechanism inside SSD-internal DRAM leveraging Processing-Using-Memory, in particular the pLUTo [65] approach. This method exploits DRAM's high storage density to enable massively parallel storage and querying of lookup tables (LUTs), ensuring efficient and scalable operations.

Querying Unit Architecture and Mechanism. Fig. 9 shows the architecture and step-by-step control flow of the Querying Unit. The hash table is stored in the SSD-internal DRAM and is queried by subsequently activating DRAM

rows using custom match logic and gated sense amplifiers (SA) [65] (highlighted in orange). The custom match logic, located adjacent to the row buffer, uses comparators to compare the currently activated row index against the key values loaded into the source row buffer. A matchline is implemented as part of the custom match logic to enable the gated SA to selectively copy the corresponding value into the output buffer, when a match is detected. A single query proceeds in four steps. ① **Key Loading.** The source row buffer is populated with the input keys (e.g., in Fig. 9: random values K, O, V). ② **Row Sweeping & Matching.** DRAM rows containing candidate hash entries are sequentially activated. For each row, the match logic compares the row index to the loaded keys. If a match is detected, the corresponding matchline is asserted. ③ **Selective Copying.** The gated sense amplifiers sense and copy only those values in the currently activated row that correspond to matched keys. ④ **Result Assembly.** The matched hash values (e.g., in Fig. 9: 6, 1, 4) are assembled in the row buffer.

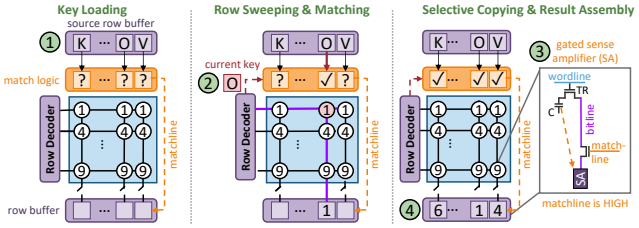

Figure 9: Overview of our hash table query mechanism in the SSD-internal DRAM.

If the DRAM size allows it, we store several copies of the hash table in the computation-enhanced subarrays to query multiple values in parallel. If the genome index exceeds DRAM capacity, MARS adopts a partitioning strategy: large indexes (e.g., 52 GB for the human genome in D5) are divided into smaller regions (e.g., 2.6 GB), which are loaded into the SSD DRAM and queried sequentially. To minimize performance impact, MARS overlaps computation with data loading, effectively hiding the data movement latency.

6.4 Chaining Implementation

Chaining (§2.1) consists of a sorting step (i.e., to sort seed positions) and a dynamic programming algorithm to extend chains from sorted seeds. While the dynamic programming part, based on additions and min operations, is efficiently handled by our near-DRAM *Arithmetic Unit*, sorting large sets of seeds directly near DRAM would be either slow or require substantial area due to custom comparator logic. Instead, we implement a highly parallel, custom sorter design inside the storage controller and benefit from increased scalability provided by the available SSD controller resources.

Key Idea. The main implementation challenge is efficiently sorting input sequences of variable length with high throughput and minimal area overhead. We address this challenge by designing a resource-efficient hierarchical mechanism consisting of (1) a *Sorter Unit* that processes input sequences of up to 128 elements and (2) a *Merger Unit* that combines smaller sorted subsequences into larger sorted outputs, enabling scalability beyond 128 elements.

Sorter and Merger Unit Architecture. MARS's Sorter and Merger Unit is based on the bitonic sorter and merger, respectively [37, 181, 197], to benefit from their inherent parallelism and hardware-friendly structure and operations. Sorter and merger units are throughput-matched to prevent pipeline stalls and are sized to balance area efficiency with maximum utilization of the available internal SSD bandwidth.

Mechanism. MARS's Control Unit manages the sorting and merging process, including data movement between the SSD-internal DRAM and the Sorter and Merger Units. Fig. 10 shows the Sort-and-Merge mechanism flow.

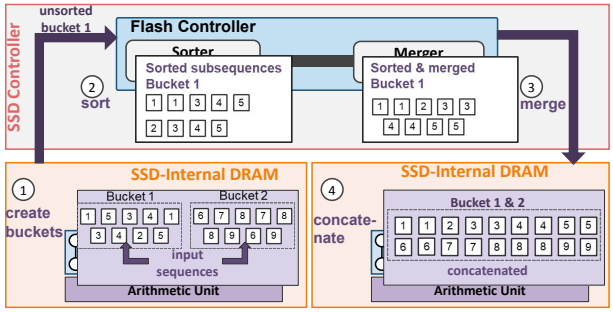

Figure 10: Simplified overview of our Sort-and-Merge workflow.

As shown in Fig. 10, ① the Contol Unit groups unsorted seeds stored in SSD-internal DRAM into eight buckets, with each bucket corresponding to a non-overlapping region of the genome. ② It transfers each bucket to one of eight parallel Sorter–Merger units located near the storage controller. Each Sorter Unit splits its assigned bucket into smaller subsequences, i.e., shorter than or equal to 128 elements, and sorts them locally using bitonic sorting. ③ If a bucket contains longer sequences, the Sorter Unit forwards the sorted subsequences to the Merger Unit. The Merger Unit then merges these short, sorted input sequences into a longer fully sorted sequence using a streaming, one-pass merge strategy with no intermediate buffering or feedback. This design enables continuous, one-pass merging with low control complexity and high throughput, especially for long or variable-length inputs. ④ The Control Unit writes the sorted outputs back to the SSD-internal DRAM. Since buckets are non-overlapping, they can be directly concatenated without further merging. If local registers near the Merger Units are insufficient, the Control Unit temporarily buffers intermediate results in DRAM.

The final sorted sequences are subsequently consumed by the dynamic programming stage of chaining.

6.5 System Integration

MARS is integrated into a modern SSD with two different modes of operation: *conventional* and *accelerator*. In *conventional mode*, the SSD operates as a storage device only. In *accelerator mode*, a MARS-enabled SSD only performs RSGA. This dual-mode of operation is feasible through small changes to the Flash Translation Layer (FTL).

MARS FTL and Data Placement. At the beginning of *accelerator* mode, the Control Unit flushes all metadata essential to the *conventional* mode (e.g., the page status table, block read counts, logical-to-physical (L2P) mapping etc.) to the flash storage. MARS leverages the access pattern of the RSGA workflow to apply a storage-efficient custom logical-to-physical (L2P) mapping for the *accelerator* mode. Since the access pattern of the genome index and reference is sequential, data is placed on the flash chips in a log-structured manner. The Control Unit then accesses the data in a sequential manner from the starting LPA and reads across channels in a round robin manner. Thus, this design allows to keep a small mapping data structure consisting of: (1) the mapping between the start logical page address (LPA) and the physical page address (PPA), (2) the database size, and (3) a sequence of physical block addresses (PBAs) rather than the complete LPA-to-PPA mappings to store the genomic data.

SSD Management Tasks. *Error Correction, Read Disturbance and Data Retention:* Since MARS 's accelerators operate within the SSD controller and the SSD-internal DRAM, all data is accessed by the Control Unit after ECC decoding [41, 42, 203, 235]. MARS effectively tackles read disturbance and data retention impact [41, 43, 44, 85, 148, 149] since: (i) The sequential access pattern of the RSGA pipeline minimizes repeated reads to the same page within short intervals, reducing the likelihood of read disturbances [43, 85], (ii) Commodity SSDs automatically apply data refresh policies that refresh pages once their read counts exceed predefined thresholds, (iii) The time interval between subsequent refreshes does not exceed the duration of RSGA, which is substantially shorter than the manufacturer-specified threshold for reliable retention age (e.g., one year [157]).

Wear-leveling: MARS effectively mitigates the impact of writes on flash lifetime thanks to two design choices: (i) Our design employs an out-of-place write policy and selects new blocks for writing based on their age, thereby effectively reducing long-term degradation, and (ii) *flash writes* are minimized as the Control Unit only writes the final read mapping results from the SSD-internal DRAM to the flash memory at the end of the RSGA workflow.

Storage Interface Commands. MARS operates independently of the host during RSGA execution, using the FSM in the SSD controller. Our design introduces two new NVMe commands, i.e., standardized interfaces used by the host to communicate with SSDs, for the host to support MARS execution: (i) *MARS_Init* initiates the RSGA analysis and signals the SSD to switch from the conventional into the accelerator mode, (ii) *MARS_Write* command updates both the MARS FTL and regular FTL at the end of the application when the read mapping results are written from the SSD-internal DRAM to flash cells.

7 Evaluation Methodology

Evaluated Systems. We evaluate MARS by comparing it against state-of-the-art RSGA and conventional basecalling-based read-mapping systems in terms of accuracy, performance and energy. As a baseline for RSGA-based read mapping, we select state-of-the-art RawHash2 [70], which offers a better accuracy-throughput trade-off compared to prior RSGA tools and techniques, including RawHash [68], Sigmap [234] and UNCALLED [119].

We evaluate the following systems: (1) **`BC`**: a baseline pipeline for basecalling-based read mapping comprised of GPU-based basecaller *Dorado* [2] and *minimap2* [128] read-mapping tool (Version 2.24-r1122). To simulate a real-time setting, we assume the basecaller processes raw signal chunks incrementally as they are generated by the sequencer rather than waiting for the full completion of each read's raw signal. (2) **`RH2`**: RawHash2 [70] RSGA-based read mapping baseline running on a state-of-the-art server-grade CPU [7]. (3) **`MS-CPU`**$_{Float}$: MARS executed on CPU using floating-point arithmetic and the filtering optimizations presented in Section 5. (4) **`MS-CPU`**$_{Fixed}$: MARS executed on CPU using both fixed-point arithmetic and filtering optimizations. (5) **`MARS`**: our proposed in-storage design of MARS using fixed-point arithmetic, implemented as described in Section 6. (6) **`MS-EXT`**: a variant of MARS that add all computation units outside (external to) the SSD. Sorting is offloaded to a near-CPU ASIC based on our custom design, while arithmetic and hash querying operations are executed in DRAM-based PIM units [65, 126]. This configuration represents a PIM-only system that avoids any in-storage computation and serves as a comparison point to evaluate the benefits of tightly integrated compute within the storage hierarchy. (7) **`MS-SIMDRAM`**: a MARS variant that replaces the Processing-Near-DRAM-based Arithmetic Unit with a SIMDRAM-based [89] Arithmetic Unit. (8) **`GenPIP`** [153]: a state-of-the-art hardware-accelerated, basecalling-based read mapping pipeline combining non-volatile memory (NVM)-based PIM with algorithmic optimizations (9) **MS-SmartSSD** [125]: an existing system [224] which directly

connects an FPGA with the SSD via an external 3 GB/s link [213]. We map MARS's Sorter and Merger Logic Units to the FPGA (300 MHz clock frequency [224]) and our PIM-components (§6.2,6.3) in the SSD-internal DRAM.

CPU and GPU Configurations. For the CPU-based systems, we use a high-end server with two 64-core AMD EPYC 7742 CPUs [7], 1TB of DDR4 DRAM [105] and a performance-optimized SSD [183] connected to the CPU via a PCIe4 interface [165]. For the BC system, the basecalling step (*Dorado* [2]) runs on an NVIDIA RTX A6000 GPU [11]. All software tools support multi-threaded processing where each raw signal sequence is handled by a separate thread. We run all tools with the best-performing configuration of 128 threads to compare against our system.

SSD and DRAM Configurations. To evaluate MARS and MS-SIMDRAM, we consider a performance-optimized SSD with internal LPDDR4 DRAM [105] (Table 1). Since accelerators and compute units operate sequentially, we simulate each component individually, including the data movement between them. For DRAM-based components (i.e., *Arithmetic* and *Querying* Units), we use timing parameters extracted from the LPDDR4 DRAM model in CACTI7 [34]. We assume that single-word ALUs embedded in SSD-internal DRAM operate at 164 MHz. For SSD components we use MQSim [78, 204], a widely-adopted simulator for modern SSDs. Our *Sorter* and *Merger* Unit are implemented in Verilog HDL and synthesized using Synopsys Design Compiler [202] at 1 GHz to obtain timing, area, and energy results. We model data movement overheads by calculating the transfer latency between each computing and storage element based on the size of the data to be transferred and the available bandwidth between components. We combine the simulation results from DRAM and SSD simulators, the Verilog synthesis and the data movement overheads to evaluate the end-to-end performance of MARS.

Table 1: Simulation configuration of our design.

Component	Detailed Configuration
SSD	NVMe, PCIe 4.0, PCIe lane BW: 1.2 GB/s, TLC, 8 channels, 8 chips/channel, tDMA: 16μs, tR (TLC): 22.5μs, flash channel BW: 1 GB/s, 4 ARM Cortex R7
SSD-Internal DRAM	4 GB LPDDR4 DRAM, 16 banks, 512 subarrays, 256 rows/subarray, row size: 2048 bytes
Sorter and Merger Unit	Frequency: 1 GHz
Arithmetic Unit	Frequency: 164 MHz

Datasets. We evaluate MARS on five real-world datasets from different organisms, covering a wide range of genome sizes. Table 2 summarizes the dataset characteristics [5, 8–10, 12, 14] and reference genomes [3, 4, 6, 13, 158, 171], all obtained from public repositories. We use the *fast5* file format for our input data and assume that the data is already correctly placed, i.e. sequentially and evenly distributed across all SSD channels, for all evaluated systems.

Table 2: Details of datasets used in our evaluation.

	Organism	Reads (#)	Bases (#)	Genome Size (bp)	Dataset Size
D1	*SARS-CoV-2*	1,382,016	594 M	29,903	11 GB
D2	*E. coli*	353,317	2,365 M	5 M	27 GB
D3	*Yeast*	49,989	380 M	12 M	39 GB
D4	*Green Algae*	29,933	609 M	111 M	74 GB
D5	*Human HG001*	269,507	1,584 M	3,117 M	39 GB

8 Evaluation

8.1 Accuracy Analysis

We evaluate **RH2**, **MS-CPU**$_{Fixed}$ and **MS-CPU**$_{Float}$ accuracy based on the ground truth generated by basecalling reads with Dorado [2] and mapping the generated *basecalled* reads to the reference genome using minimap2 [128]. All hardware systems implement MS-CPU$_{Float}$ workflow and thus achieve the same accuracy. We use UNCALLED paftats [119] tool to identify true positives (TP: correct mappings), false positives (FP: incorrect mappings), and false negatives (FN: unmapped reads that are mapped in the ground truth) based on the mapping position distance from the respective ground truth. Using these values, we calculate precision (P = TP/(TP+FP)), recall (R = TP/(TP+FN)), and the F_1 score (F_1 = 2x(PxR)/(P+R)).

We make two observations based on the accuracy results reported in Table 3. (1) MS-CPU$_{Fixed}$ outperforms RH2 in terms of recall and F_1 score for all evaluated datasets, while maintaining on-par precision for small genomes and only a slight reduction in precision for larger genomes. This improvement is due to the integration of our two proposed filtering techniques (§5.1) and early quantization (§5.2) which together eliminate ambiguous or redundant candidate matches, i.e., matches that are frequent, low-quality or non-specific, and allow the pipeline to focus on signal regions that are more likely to represent correct alignments. (2) The use of fixed-point and integer operations instead of floating-point operations only minimally decreases accuracy for all datasets.

8.2 Performance Analysis

We evaluate the performance of all seven systems described in §7 leveraging the five diverse datasets of Table 2. Fig. 11 shows the execution time speedup achieved by each evaluated system over CPU-based RawHash2 RH2. We make three observations. First, MARS outperforms *all* other baselines across *all* datasets. Compared to the GPU-accelerated basecalling based pipeline BC, MARS delivers a speedup of 93× on average across all five datasets with larger speedups for smaller genomes. This is because MARS (i) eliminates basecalling, (ii) applies filtering mechanisms, (iii) reduces data movement, and (iv) enables highly parallel in-storage execution.

Second, MARS outperforms all prior hardware-accelerated solutions: MS-EXT, MS-SIMDRAM, GenPIP, and MS-SmartSSD.

Table 3: Mapping accuracy of three RSGA pipelines compared to basecalling-based ground truth.

	D1 *SARS-CoV-2*			D2 *E.coli*			D3 *Yeast*			D4 *Green Algae*			D5 *Human HG001*		
	Prec.	Recall	F_1	Prec.	Recall	F_1	Prec.	Recall	F_1	Prec.	Recall	F_1	Prec.	Recall	F_1
RH2	0.9868	0.8735	0.9267	0.9573	0.9009	0.9282	0.9862	0.8412	0.9079	0.9691	0.7015	0.8139	0.8949	0.4054	0.5582
MS-CPU$_{Fixed}$	0.9917	0.9694	0.9803	0.9854	0.9574	0.9712	0.9533	0.9643	0.9588	0.9125	0.9166	0.9141	0.8723	0.6318	0.7300
MS-CPU$_{Float}$	0.9939	0.9796	0.9867	0.9893	0.9616	0.9753	0.9551	0.9655	0.9603	0.9254	0.9438	0.9354	0.8763	0.6729	0.7612

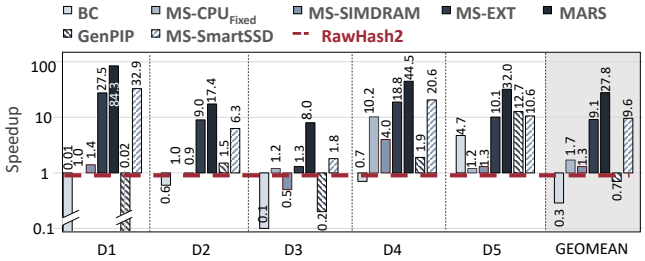

Figure 11: End-to-end execution time speedup of each system over RH2.

Specifically, MARS improves performance by 3.1× on average over MS-EXT, which adopts PIM solutions (MARS-based ASIC and PIM accelerator) outside the storage. This comparison point shows that MS-EXT fails to fundamentally solve the I/O data movement overhead problem and highlights the importance and need for in-storage processing for RSGA. MS-SmartSSD performs worse than MARS, due to its limited 3 GB/s bandwidth between SSD and FPGA [125], which restricts the use of internal SSD bandwidth between flash and storage controller, fully utilized by MARS. While MS-SIMDRAM addresses I/O overhead through in-storage computation, its use of bit-serial operations for arithmetic (e.g., multiplication, division) results in execution time 21.4× slower than MARS. MARS is the only design that both eliminates the I/O bottleneck and meets the computational demands of RSGA acceleration.

Third, our algorithmic improvements alone (MS-CPU$_{Fixed}$), i.e., without leveraging ISP capabilities, provide a considerable speedup of 1.2-10.2× over RH2 for medium- to large-sized genomes (i.e., D3-D5) and on-par performance for small genomes (i.e., D1,D2). This demonstrates the effectiveness of our software optimizations, including filtering, in reducing the computational load, particularly during chaining.

Throughput evaluation. We compare MARS's throughput with the throughput of a single sequencer, which is 450 bases per second (i.e., 4000 - 5000 samples per second) [216]. As Table 4 shows, MARS's throughput is substantially higher than 450 bp/sec for all datasets. In fact, MARS outperforms the real-time analysis requirement of a full MinION sequencer [103], which processes data at 230,400 bp/s, by 46× on average across all datasets (between 1.2× for large genomes (D5) to 202× for small genomes (D1)).

Table 4: Throughput of MARS. A single nanopore has a throughput of 450 bp/sec; an entire MinION sequencer achieves 230,400 bp/sec.

	D1	D2	D3	D4	D5
Throughput [bp/sec]	46,655,128	5,274,148	1,202,660	1,277,764	286,728

8.3 Energy Analysis

To demonstrate the energy benefits of MARS, we measure the energy consumption of all components (i.e., SSD, DRAM, CPU and if applicable GPU) involved in the respective systems. We use AMD μProf [28] to measure the energy consumption for CPU-based systems, and the CACTI7 [34] DDR4 model to estimate the power overheads on our PIM-enabled DRAM design. We synthesize logic components with the Synopsys Design Compiler [202] using a 65nm process node to estimate their power consumption.

Fig. 12 shows the end-to-end energy reduction achieved by all evaluated systems over RawHash2 (RH2). We make three observations. (1) All hardware-accelerated systems, i.e., MARS, MS-EXT, MS-SIMDRAM, GenPIP achieve greater energy reduction compared to CPU-based setups, i.e., BC and MS-CPU$_{Fixed}$. (2) Only MS-SIMDRAM yields higher energy reduction compared to MARS (by 3.5× on average across datasets), due to its simplified Arithmetic Unit based on bit-serial, in-memory execution. However, because of MS-SIMDRAM's significantly higher latency (§8.2), MARS still provides a more favorable trade-off between latency and energy consumption. (3) MS-EXT reduces energy by 22.3× as opposed to MARS's 79.4× reduction over RH2, due to high data movement from the storage to the host and accelerators and a greater reliance on the CPU for orchestration, which increases energy use on the host side. Overall, MARS achieves the best energy consumption and performance trade-off among all designs.

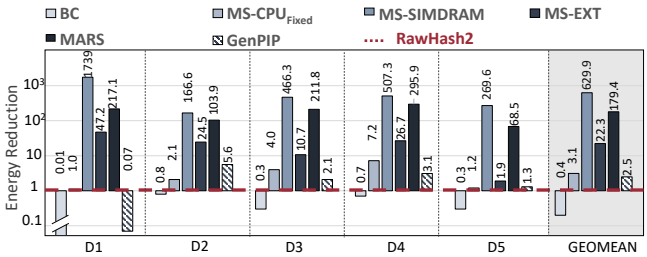

Figure 12: Energy reduction of each system compared to RH2.

8.4 Area Analysis

SSD-internal DRAM overhead. We estimate the base area of our PIM-enabled DRAM with CACTI7 [34] to be 55.48 mm^2 in a 22nm technology. Each Arithmetic Unit occupies 0.0295 mm^2, leading to 7.56 mm^2 [126, 127] total overhead for all 256 Arithmetic Units. Each LUT-based Querying Unit occupies 0.018 mm^2, leading to 9.22 mm^2 [65] for 512 instances. The total DRAM overhead of our design, i.e., 16.78 mm^2, is low compared to the total SSD area available, i.e., at least 6400 mm^2 for our SSD configuration of 8 channels and 8 typical 100 mm^2 NAND flash chips per channel.

SSD Controller Logic overhead. We estimate the area overhead of our logic components using Synopsys Design Compiler [202] with UMC 65nm technology node [208]. The area for the Sorter, Merger and Controller Unit is 0.78 mm^2, 0.14 mm^2 and 0.002 mm^2, respectively. Compared to a 14nm Intel Processor [218], the Sorter and Merger introduce only 0.028% area overhead (the area is 0.09 mm^2 when scaled to 14nm [199]).

Table 5: Area analysis overview per component.

Placement in SSD	Unit	Instances Number	Area [mm^2] per Unit	Area [mm^2] Total
SSD-internal DRAM	Arithmetic	256	0.0295	7.56
	Querying	512	0.018	9.22
SSD controller	Sorter	8	0.78	6.24
	Merger	8	0.14	1.12
	Control	1	0.002	0.002

8.5 Sensitivity to SSD-Internal DRAM Size

We perform a sensitivity analysis to examine the scalability of ISP designs MARS and MS-SIMDRAM for different sizes of the SSD-internal DRAM, i.e., 2 GB, 4GB (base configuration) and 8 GB. Fig. 13 shows that MARS's performance increases by 1.70x on average when we double the internal DRAM size, while MS-SIMDRAM's performance increases almost by 1.99x on average. Therefore, the proposed design scales well when increasing internal DRAM resources and is not bound by the internal bandwidth. MS-SIMDRAM's slightly better scaling indicates that increasing the DRAM capacity yields better results for PuM-based computations.

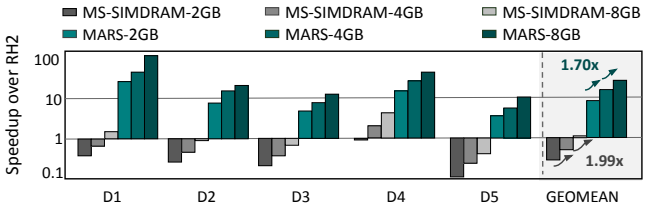

Figure 13: Sensitivity to SSD-internal DRAM size.

9 Related Work

To our knowledge, this is the first work to 1) enable in-storage acceleration of Raw Signal Genome Analysis and 2) combine the use of processing near memory and processing using memory inside the storage system. In this section, we briefly review prior work on hardware acceleration for genome analysis and ISP.

Hardware Acceleration for RSGA. Prior hardware acceleration works on RSGA propose FPGA-based [141, 180, 190, 200] and GPU-based [36, 74, 82, 178] systems. Specifically, Squigglefilter [60] proposes an edge-GPU-based system for RSGA that performs contamination analysis for small, viral genomes based on a 1D systolic array. HARU [190] uses an MPSoC with an on-chip FPGA to accelerate RSGA and f5c [74] presents a GPU-based accelerator. None of these systems 1) consider the impact of I/O data movement on end-to-end execution of RSGA and 2) provide a system for genome analysis that is scalable to medium- and larger-sized genomes, due to the use of costly dynamic time-warping alignment operations [60, 135, 184]. A comparison of MARS with SquiggleFilter and HARU is out of scope, as these works focus on performing read alignment for small, mostly viral genomes. MARS can be integrated into these tools to help them quickly identify seed hits, thus avoid searching the entire genome and enable scaling to large genomes.

Hardware Acceleration for Genome Analysis. Multiple prior works propose accelerator designs for basecalling-based genome analysis targeting basecalling and read mapping steps with different architectures like ASICs [72, 150, 207], GPUs [17, 51, 57, 82, 94, 95, 137–140, 162, 219, 231], FPGAs [35, 47–50, 64, 73, 76, 88, 90, 91, 130, 143, 176, 209, 221], ISP [152], and PIM [30, 84, 98, 114, 131, 185, 233, 238]. Basecalling accelerators [32, 145, 146, 153, 188, 195, 222, 223, 228] speed up the translation of raw signals into nucleotide sequences, a step that is entirely bypassed by our RSGA-based design. Read mapping accelerators [31, 40, 47, 48, 50, 51, 59, 80, 81, 87, 94, 97, 101, 111, 112, 114, 121, 143, 152, 167, 236] are *not* applicable to RSGA as they do *not* consider the noise within raw signals.

In-Storage-Processing. Prior works explore ISP through various approaches using (1) Processing-Near-Flash memory by integrating processing capabilities into the SSD controller in a general-purpose [15, 79, 110, 113, 212, 240] or application-specific way [58, 107, 117, 151, 166, 172, 173, 187, 214], (2) Processing-using-Flash memory by exploiting the analog properties of flash memory [56, 75, 92, 147, 154, 163, 191, 206, 211] or by (3) closely integrating SSDs with GPUs [54] or FPGAs [18, 106, 118, 205] (e.g., SmartSSD [125, 213]). While SmartSSD [125] places an FPGA near the SSD, MARS integrates computation inside the SSD-internal DRAM and controllers. Several works also consider other storage technologies like HDDs [55, 113, 172, 173] for

computation. None of these works leverages SSD's computational capabilities and enhances them to accelerate RSGA.

10 Conclusion

We propose MARS, the first in-storage processing architecture that enables multiple Processing-In-Memory paradigms within the SSD to reduce both data movement and computation overheads of RSGA read mapping. MARS (1) proposes targeted software modifications, such as early signal quantization and read filtering, to minimize hardware resources while maintaining accuracy, and (2) provides near-memory computation units within the SSD for accelerating computational steps of the RSGA pipeline. MARS improves performance over software and hardware-accelerated state-of-the-art read mapping pipelines by a factor of 93× and 40× while reducing their energy consumption by 427× and 72× on average across five real-world dataset.

Acknowledgments

We thank the anonymous reviewers of ICS 2025, ISCA 2025 and HPCA 2025 for their feedback. We thank the SAFARI group members for the feedback and stimulating intellectual environment they provide. We acknowledge the generous gifts from our industrial partners including Google, Huawei, Intel, and Microsoft. This work is supported in part by the ETH Future Computing Laboratory (EFCL), Huawei ZRC Storage Team, Semiconductor Research Corporation, AI Chip Center for Emerging Smart Systems (ACCESS), sponsored by InnoHK funding, Hong Kong SAR, and European Union's Horizon programme for research and innovation [101047160 - BioPIM].

References

[1] [n. d.]. Bonito. https://github.com/nanoporetech/bonito.
[2] [n. d.]. Dorado. https://github.com/nanoporetech/dorado.
[3] 2002. Escherichia coli Reference Genome GCA_000007445.1. https://ftp.ncbi.nlm.nih.gov/genomes/all/GCA/000/007/445/GCA_000007445.1_ASM744v1/GCA_000007445.1_ASM744v1_genomic.fna.gz.
[4] 2014. Yeast Reference Genome GCA_000146045.2. https://hgdownload.soe.ucsc.edu/goldenPath/sacCer3/bigZips/sacCer3.fa.gz.
[5] 2017. Human Genome Dataset, SRA Accession: FAB42260. http://s3.amazonaws.com/nanopore-human-wgs/rel6/MultiFast5Tars/FAB42260-4177064552_Multi_Fast5.tar.
[6] 2018. Green Algae Reference Genome GCF_000002595.2. https://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/000/002/595/GCF_000002595.2_Chlamydomonas_reinhardtii_v5.5/GCF_000002595.2_Chlamydomonas_reinhardtii_v5.5_genomic.fna.gz.
[7] 2019. AMD® EPYC® 7742 CPU. https://www.amd.com/en/products/cpu/amd-epyc-7742.
[8] 2019. Green Algae Genome Dataset, SRA Accession: ERR3237140. https://sra-pub-src-2.s3.amazonaws.com/ERR3237140/Chlamydomonas_0.tar.gz.
[9] 2019. Oxford Nanopore Human Reference Datasets. https://github.com/nanopore-wgs-consortium/NA12878.
[10] 2019. Yeast Genome Dataset, SRA Accession: SRR8648503. https://sra-pub-src-1.s3.amazonaws.com/SRR8648503/GLU1II_basecalled_fast5_1.tar.gz.
[11] 2020. NVIDIA RTX A6000. https://www.nvidia.com/en-us/design-visualization/rtx-a6000/.
[12] 2020. SARS-Cov-2 Genome Dataset. https://cadde.s3.climb.ac.uk/SP1-raw.tgz.
[13] 2020. SARS-Cov-2 Reference Genome GCF_009858895.2. https://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/009/858/895/GCF_009858895.2_ASM985889v3/GCF_009858895.2_ASM985889v3_genomic.fna.gz.
[14] 2021. Escherichia coli Genome Dataset, SRA Accession: ERR9127551. https://sra-pub-src-2.s3.amazonaws.com/ERR9127551/ecoli_r9.tar.gz.
[15] Anurag Acharya, Mustafa Uysal, and Joel Saltz. 1998. Active Disks: Programming Model, Algorithms and Evaluation. *ASPLOS* (1998).
[16] Nitin Agrawal, Vijayan Prabhakaran, Ted Wobber, John D Davis, Mark Manasse, and Rina Panigrahy. 2008. Design Tradeoffs for SSD Performance. In *USENIX ATC*.
[17] Nauman Ahmed, Jonathan Levy, Shanshan Ren, Hamid Mushtaq, Koen Bertels, and Zaid Al-Ars. 2019. GASAL2: a GPU Accelerated Sequence Alignment Library for High-throughput NGS data. *BMC bioinformatics* (2019).
[18] Mohammadamin Ajdari, Pyeongsu Park, Joonsung Kim, Dongup Kwon, and Jangwoo Kim. 2019. CIDR: A Cost-effective in-line Data Reduction System for terabit-per-second Scale SSD Arrays. In *HPCA*.
[19] Alexandra Sneddon, Agin Ravindran, Nadine Hein, Nikolay Shirokikh, and Eduardo Eyras. 2022. Real-time Biochemical-free Targeted Sequencing of RNA species with RISER. *bioRxiv* (2022).
[20] Can Alkan, Jeffrey M Kidd, Tomas Marques-Bonet, Gozde Aksay, Francesca Antonacci, Fereydoun Hormozdiari, Jacob O Kitzman, Carl Baker, Maika Malig, Onur Mutlu, S Cenk Sahinalp, Richard A Gibbs, and Evan E Eichler. 2009. Personalized Copy Number and Segmental Duplication Maps Using Next-Generation Sequencing. *Nature Genetics* (2009).
[21] Mohammed Alser, Zülal Bingöl, Damla Senol Cali, Jeremie Kim, Saugata Ghose, Can Alkan, and Onur Mutlu. 2020. Accelerating Genome Analysis: A Primer on An Ongoing Journey. *Micro* (2020).
[22] Mohammed Alser, Hasan Hassan, Akash Kumar, Onur Mutlu, and Can Alkan. 2019. Shouji: A Fast and Efficient Pre-alignment Filter for Sequence Alignment. *Bioinformatics* (2019).
[23] Mohammed Alser, Hasan Hassan, Hongyi Xin, Onur Ergin, Onur Mutlu, and Can Alkan. 2017. GateKeeper: a new Hardware Architecture for Accelerating Pre-alignment in DNA Short Read Mapping. *Bioinformatics* (2017).
[24] Mohammed Alser, Joel Lindegger, Can Firtina, Nour Almadhoun, Haiyu Mao, Gagandeep Singh, Juan Gomez-Luna, and Onur Mutlu. 2022. From Molecules to Genomic Variations: Accelerating Genome Analysis via Intelligent Algorithms and Architectures. *CSBJ* (2022).
[25] Muhammed Alser, Onur Mutlu, and Can Alkan. 2017. MAGNET: Understanding and improving the accuracy of genome pre-Alignment filtering.
[26] Mohammed Alser, Jeremy Rotman, Kodi Taraszka, Huwenbo Shi, Pelin Icer Baykal, Harry Taegyun Yang, Victor Xue, Sergey Knyazev, Benjamin D Singer, Brunilda Balliu, et al. 2021. Technology Dictates Algorithms: Recent Developments in Read Alignment. *Genome Biology* (2021).
[27] Mohammed Alser, Taha Shahroodi, Juan Gómez-Luna, Can Alkan, and Onur Mutlu. 2020. SneakySnake: A Fast and Accurate Universal Genome Pre-alignment Filter for CPUs, GPUs and FPGAs. *Bioinformatics* (2020).
[28] AMD. [n. d.]. AMD μProf. https://www.amd.com/en/developer/uprof.html.

[29] AnandTech. 2020. New Enterprise SSD Controllers. https://www.anandtech.com/show/16275/new-enterprise-ssd-controllers-from-silicon-motion-phison-fadu.
[30] Shaahin Angizi, Jiao Sun, Wei Zhang, and Deliang Fan. 2019. Aligns: A Processing-in-Memory Accelerator for DNA Short Read Alignment leveraging sot-mram. In *DAC*.
[31] Shaahin Angizi, Jiao Sun, Wei Zhang, and Deliang Fan. 2020. PIM-Aligner: A Processing-in-MRAM Platform for Biological Sequence Alignment. In *DATE*.
[32] Aayush Ankit, Izzat El Hajj, Sai Rahul Chalamalasetti, Geoffrey Ndu, Martin Foltin, R. Stanley Williams, Paolo Faraboschi, Wen mei Hwu, John Paul Strachan, Kaushik Roy, and Dejan S Milojicic. 2019. PUMA: A Programmable Ultra-efficient Memristor-based Accelerator for Machine Learning Inference.
[33] Euan A. Ashley. 2016. Towards Precision Medicine. *Nature Reviews Genetics* (2016).
[34] Rajeev Balasubramonian, Andrew B. Kahng, Naveen Muralimanohar, Ali Shafiee, and Vaishnav Srinivas. 2017. CACTI 7: New Tools for Interconnect Exploration in Innovative Off-Chip Memories. *ACM Trans. Archit. Code Optim.* (2017).
[35] Subho Sankar Banerjee, Mohamed El-Hadedy, Jong Bin Lim, Zbigniew T Kalbarczyk, Deming Chen, Steven S Lumetta, and Ravishankar K Iyer. 2018. ASAP: Accelerated Short-Read Alignment on Programmable Hardware. *TC* (2018).
[36] Yuwei Bao, Jack Wadden, John R. Erb-Downward, Piyush Ranjan, Weichen Zhou, Torrin L. McDonald, Ryan E. Mills, Alan P. Boyle, Robert P. Dickson, David Blaauw, and Joshua D. Welch. 2021. SquiggleNet: Real-time, Direct Classification of Nanopore Signals. *Genome Biology* (2021).
[37] Kenneth E Batcher. 1968. Sorting Networks and their Applications. In *AFIPS*.
[38] Zülal Bingöl, Mohammed Alser, Onur Mutlu, Ozcan Ozturk, and Can Alkan. 2021. GateKeeper-GPU: Fast and Accurate Pre-Alignment Filtering in Short Read Mapping. *IPDPSW* (2021).
[39] Joshua S Bloom, Laila Sathe, Chetan Munugala, Eric M Jones, Molly Gasperini, Nathan B Lubock, Fauna Yarza, Erin M Thompson, Kyle M Kovary, Jimin Park, et al. 2021. 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).
[40] Luk Burchard, Max Xiaohang Zhao, Johannes Langguth, Aydın Buluç, and Giulia Guidi. 2023. Space Efficient Sequence Alignment for SRAM-Based Computing: X-Drop on the Graphcore IPU. *SC* (2023).
[41] Yu Cai, Saugata Ghose, Erich F Haratsch, Yixin Luo, and Onur Mutlu. 2017. Error Characterization, Mitigation, and Recovery in Flash-Memory-based Solid-State Drives. *Proc. IEEE* (2017).
[42] Yu Cai, Saugata Ghose, Erich F Haratsch, Yixin Luo, and Onur Mutlu. 2018. Errors in Flash-Memory-based Solid-State Drives: Analysis, Mitigation, and Recovery. *Inside Solid State Drives* (2018).
[43] Yu Cai, Yixin Luo, Saugata Ghose, and Onur Mutlu. 2015. Read Disturb Errors in MLC NAND Flash Memory: Characterization, Mitigation, and Recovery. In *DSN*.
[44] Yu Cai, Yixin Luo, Erich F Haratsch, Ken Mai, and Onur Mutlu. 2015. Data Retention in MLC NAND Flash Memory: Characterization, Optimization, and Recovery. In *HPCA*.
[45] Meryem Banu Cavlak, Gagandeep Singh, Mohammed Alser, Can Firtina, Joel Lindegger, Mohammad Sadrosadati, Nika Mansouri Ghiasi, Can Alkan, and Onur Mutlu. 2024. TargetCall: Eliminating the Wasted Computation in Basecalling via Pre-Basecalling Filtering. *Frontiers in Genetics* (2024).
[46] Fan Chen, Linghao Song, Yiran Chen, et al. 2020. PARC: A Processing-In-CAM Architecture for Genomic Long Read Pairwise Alignment using ReRAM. In *ASP-DAC*.
[47] Peng Chen, Chao Wang, Xi Li, and Xuehai Zhou. 2014. Accelerating the Next Generation Long Read Mapping with the FPGA-based System. *TCBB* (2014).
[48] Yen-Lung Chen, Bo-Yi Chang, Chia-Hsiang Yang, and Tzi-Dar Chiueh. 2021. A High-Throughput FPGA Accelerator for Short-Read Mapping of the Whole Human Genome. *TPDS* (2021).
[49] Yu-Ting Chen, Jason Cong, Zhenman Fang, Jie Lei, and Peng Wei. 2016. When Spark Meets FPGAs: A Case Study for Next-Generation DNA Sequencing Acceleration. In *HotCloud*.
[50] Yu-Ting Chen, Jason Cong, Jie Lei, and Peng Wei. 2015. A Novel High-Throughput Acceleration Engine for Read Alignment. In *FCCM*.
[51] Haoyu Cheng, Yong Zhang, and Yun Xu. 2018. Bitmapper2: A GPU-accelerated All-Mapper based on the Sparse q-gram Index. *TCBB* (2018).
[52] Yiran Cheng, Xibo Sun, and Qiong Luo. 2024. RapidGKC: GPU-Accelerated K-Mer Counting. In *ICDE*.
[53] G. M. Cherf, K. R. Lieberman, H. Rashid, C. E. Lam, K. Karplus, and M. Akeson. 2012. Automated Forward and Reverse Ratcheting of DNA in a Nanopore at 5-Å Precision. *Nature Biotechnology* (2012).
[54] Benjamin Y Cho, Won Seob Jeong, Doohwan Oh, and Won Woo Ro. 2013. Xsd: Accelerating Mapreduce by Harnessing the GPU Inside an SSD. In *WoNDP*.
[55] Sangyeun Cho, Chanik Park, Hyunok Oh, Sungchan Kim, Youngmin Yi, and Gregory R Ganger. 2013. Active Disk Meets Flash: A Case for Intelligent SSDs. In *ICS*.
[56] Won Ho Choi, Pi-Feng Chiu, Wen Ma, Gertjan Hemink, Tung Thanh Hoang, Martin Lueker-Boden, and Zvonimir Bandic. 2020. An In-Flash Binary Neural Network Accelerator with SLC NAND Flash Array. In *ISCAS*.
[57] Edans Flavius de Oliveira Sandes, Guillermo Miranda, Xavier Martorell, Eduard Ayguade, George Teodoro, and Alba Cristina Magalhaes Melo. 2016. CUDAlign 4.0: Incremental Speculative Traceback for Exact Chromosome-wide Alignment in GPU Clusters. *TPDS* (2016).
[58] Jaeyoung Do, Yang-Suk Kee, Jignesh M Patel, Chanik Park, Kwanghyun Park, and David J DeWitt. 2013. Query Processing on Smart SSDs: Opportunities and Challenges. In *SIGMOD*.
[59] Max Doblas, Oscar Lostes-Cazorla, Quim Aguado-Puig, Nicholas Cebry, Pau Fontova-Musté, Christopher Batten, Santiago Marco-Sola, and Miquel Moretó. 2023. GMX: Instruction Set Extensions for Fast, Scalable, and Efficient Genome Sequence Alignment. In *MICRO*.
[60] Tim Dunn, Harisankar Sadasivan, Jack Wadden, Kush Goliya, Kuan-Yu Chen, David Blaauw, Reetuparna Das, and Satish Narayanasamy. 2021. SquiggleFilter: An Accelerator for Portable Virus Detection. In *MICRO*.
[61] Harrison S. Edwards, Raga Krishnakumar, Anupama Sinha, Sara W. Bird, Kamlesh D. Patel, and Michael S. Bartsch. 2019. Real-Time Selective Sequencing with RUBRIC: Read Until with Basecall and Reference-Informed Criteria. *Sci. Rep.* (2019).
[62] Hans Ellegren and Nicolas Galtier. 2016. Determinants of Genetic Diversity. *Nature Reviews Genetics* (2016).
[63] Kam Eshghi and Rino Micheloni. 2018. SSD Architecture and PCI Express Interface. In *SSDs*.
[64] Xia Fei, Zou Dan, Lu Lina, Man Xin, and Zhang Chunlei. 2018. FPGASW: Accelerating Large-scale Smith–Waterman Sequence Alignment Application with Backtracking on FPGA Linear Systolic Array. *Interdisciplinary Sciences: Computational Life Sciences* (2018).
[65] João Dinis Ferreira, Gabriel Falcao, Juan Gómez-Luna, Mohammed Alser, Lois Orosa, Mohammad Sadrosadati, Jeremie S. Kim, Geraldo F. Oliveira, Taha Shahroodi, Anant Nori, and Onur Mutlu. 2022. pLUTo: Enabling Massively Parallel Computation in DRAM via Lookup Tables. In *MICRO*.

[66] Can Firtina. 2025. Enabling Fast, Accurate, and Efficient Real-Time Genome Analysis via New Algorithms and Techniques. *arXiv preprint arXiv:2503.02997* (2025).

[67] Can Firtina and Can Alkan. 2016. On Genomic Repeats and Reproducibility. *Bioinformatics* (2016).

[68] Can Firtina, Nika Mansouri Ghiasi, Joel Lindegger, Gagandeep Singh, Meryem Banu Cavlak, Haiyu Mao, and Onur Mutlu. 2023. RawHash: Enabling Fast and Accurate Real-Time Analysis of Raw Nanopore Signals for Large Genomes. *Bioinformatics* (2023).

[69] Can Firtina, Maximilian Mordig, Harun Mustafa, Sayan Goswami, Nika Mansouri Ghiasi, Stefano Mercogliano, Furkan Eris, Joël Lindegger, Andre Kahles, and Onur Mutlu. 2024. Rawsamble: Overlapping and Assembling Raw Nanopore Signals using a Hash-based Seeding Mechanism. *arXiv* (2024).

[70] Can Firtina, Melina Soysal, Joël Lindegger, and Onur Mutlu. 2024. RawHash2: Mapping Raw Nanopore Signals Using Hash-Based Seeding and Adaptive Quantization. *Bioinformatics* (2024).

[71] Mauricio Flores, Gustavo Glusman, Kristin Brogaard, Nathan D Price, and Leroy Hood. 2013. P4 Medicine: How Systems Medicine Will Transform the Healthcare Sector and Society. *Personalized Medicine* (2013).

[72] Daichi Fujiki, Arun Subramaniyan, Tianjun Zhang, Yu Zeng, Reetuparna Das, David Blaauw, and Satish Narayanasamy. 2018. GenAx: A Genome Sequencing Accelerator. In *ISCA*.

[73] Daichi Fujiki, Shunhao Wu, Nathan Ozog, Kush Goliya, David Blaauw, Satish Narayanasamy, and Reetuparna Das. 2020. SeedEx: A Genome Sequencing Accelerator for Optimal Alignments in Subminimal Space. In *MICRO*.

[74] Hasindu Gamaarachchi, Chun Wai Lam, Gihan Jayatilaka, Hiruna Samarakoon, Jared T. Simpson, Martin A. Smith, and Sri Parameswaran. 2020. GPU Accelerated Adaptive Banded Event Alignment for Rapid Comparative Nanopore Signal Analysis. *BMC Bioinformatics* (2020).

[75] Congming Gao, Xin Xin, Youyou Lu, Youtao Zhang, Jun Yang, and Jiwu Shu. 2021. ParaBit: Processing Parallel Bitwise Operations in NAND Flash Memory Based SSDs. In *MICRO*.

[76] Amit Goyal, Hyuk Jung Kwon, Kichan Lee, Reena Garg, Seon Young Yun, Yoon Hee Kim, Sunghoon Lee, and Min Seob Lee. 2017. Ultra-fast Next Generation Human Genome Sequencing Data Processing using DRAGENTM bio-IT Processor for Precision Medicine. *Open Journal of Genetics* (2017).

[77] Alexander L. Greninger, Samia N. Naccache, Scot Federman, Guixia Yu, Placide Mbala, Vanessa Bres, Doug Stryke, Jerome Bouquet, Sneha Somasekar, Jeffrey M. Linnen, Roger Dodd, Prime Mulembakani, Bradley S. Schneider, Jean-Jacques Muyembe-Tamfum, Susan L. Stramer, and Charles Y. Chiu. 2015. Rapid Metagenomic Identification of Viral Pathogens in Clinical Samples by Real-time Nanopore Sequencing Analysis. *Genome Medicine* (2015).

[78] CMU SAFARI Research Group. 2018. MQSim: A Framework for SSD Simulation - GitHub. https://github.com/CMU-SAFARI/MQSim.

[79] Boncheol Gu, Andre S. Yoon, Duck-Ho Bae, Insoon Jo, Jinyoung Lee, Jonghyun Yoon, Jeong-Uk Kang, Moonsang Kwon, Chanho Yoon, Sangyeun Cho, Jaeheon Jeong, and Duckhyun Chang. 2016. Biscuit: A Framework for near-Data Processing of Big Data Workloads. In *ISCA*.

[80] Yufeng Gu, Arun Subramaniyan, Tim Dunn, Alireza Khadem, Kuan-Yu Chen, Somnath Paul, Md Vasimuddin, Sanchit Misra, David Blaauw, Satish Narayanasamy, et al. 2023. GenDP: A Framework of Dynamic Programming Acceleration for Genome Sequencing Analysis. In *ISCA*.

[81] Venkateshwarlu Yellaswamy Gudur, Sidharth Maheshwari, Amit Acharyya, and Rishad Shafik. 2021. An FPGA based Energy-Efficient Read Mapper with Parallel Filtering and In-Situ Verification. *TCBB* (2021).

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

[83] Aayush Gupta, Youngjae Kim, and Bhuvan Urgaonkar. 2009. DFTL: A Flash Translation Layer Employing Demand-based Selective Caching of Page-level Address Mappings. In *ASPLOS*.

[84] Saransh Gupta, Mohsen Imani, Behnam Khaleghi, Venkatesh Kumar, and Tajana Rosing. 2019. RAPID: A ReRAM Processing In-Memory Architecture for DNA Sequence Alignment. In *ISLPED*.

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

[86] Faraz Hach, Iman Sarrafi, Farhad Hormozdiari, Can Alkan, Evan E Eichler, and S Cenk Sahinalp. 2014. mrsFAST-Ultra: A Compact, SNP-Aware Mapper for High Performance Sequencing Applications. *Nucleic acids research* (2014).

[87] Abbas Haghi, Lluc Alvarez, Jordi Front, Juan Miguel De Haro Ruiz, Roger Figueras, Max Doblas, Santiago Marco-Sola, and Miquel Moreto. 2023. WFAsic: A High-Performance ASIC Accelerator for DNA Sequence Alignment on a RISC-V SoC. In *ICPP*.

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

[89] Nastaran Hajinazar, Geraldo F Oliveira, Sven Gregorio, João Dinis Ferreira, Nika Mansouri Ghiasi, Minesh Patel, Mohammed Alser, Saugata Ghose, Juan Gómez-Luna, and Onur Mutlu. 2021. SIMDRAM: A Framework for Bit-Serial SIMD Processing Using DRAM. In *ASPLOS*.

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

[91] Tae Jun Ham, Yejin Lee, Seong Hoon Seo, U Gyeong Song, Jae W Lee, David Bruns-Smith, Brendan Sweeney, Krste Asanovic, Young H Oh, and Lisa Wu Wills. 2021. Accelerating Genomic Data Analytics with Composable Hardware Acceleration Framework. *Micro* (2021).

[92] Runze Han, Peng Huang, Yachen Xiang, Chen Liu, Zhen Dong, Zhiqiang Su, Yongbo Liu, Lu Liu, Xiaoyan Liu, and Jinfeng Kang. 2019. A Novel Convolution Computing Paradigm based on NOR Flash Array with High Computing Speed and Energy Efficiency. *TCAS I* (2019).

[93] Seunghee Han, Seungjae Moon, Teokkyu Suh, JaeHoon Heo, and Joo-Young Kim. 2024. BLESS: Bandwidth and Locality Enhanced SMEM Seeding Acceleration for DNA Sequencing. In *ISCA*.

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

[95] Ernst Joachim Houtgast, Vlad-Mihai Sima, Koen Bertels, and Zaid Al-Ars. 2018. Hardware Acceleration of BWA-MEM Genomic Short Read Mapping for Longer Read Lengths. *Computational biology and chemistry* (2018).

[96] Neng Huang, Fan Nie, Peng Ni, Feng Luo, and Jianxin Wang. 2020. SACall: A Neural Network Basecaller for Oxford Nanopore Sequencing Data Based on Self-Attention Mechanism. *TCBB* (2020).

[97] Yi Huang, Lingkun Kong, Dibei Chen, Zhiyu Chen, Xiangyu Kong, Jianfeng Zhu, Konstantinos Mamouras, Shaojun Wei, Kaiyuan Yang, and Leibo Liu. 2023. CASA: An Energy-Efficient and High-Speed CAM-based SMEM Seeding Accelerator for Genome Alignment. In

MICRO.

[98] Wenqin Huangfu, Shuangchen Li, Xing Hu, and Yuan Xie. 2018. RADAR: A 3D-ReRAM based DNA Alignment Accelerator Architecture. In *DAC*.

[99] W. Huangfu, X. Li, S. Li, X. Hu, P. Gu, and Y. Xie. 2019. MEDAL: Scalable DIMM-based Near Data Processing Accelerator for DNA Seeding Algorithm. In *MICRO*.

[100] W. Huangfu, K. T. Malladi, S. Li, P. Gu, and Y. Xie. 2020. NEST: DIMM-based Near-Data-Processing Accelerator for K-mer Counting. In *ICCAD*.

[101] Zuher Jahshan, Itay Merlin, Esteban Garzón, and Leonid Yavits. 2023. DASH-CAM: Dynamic Approximate SearcH Content Addressable Memory for genome classification. *MICRO* (2023).

[102] Z. Jahshan and L. Yavits. 2024. MajorK: Majority Based kmer Matching in Commodity DRAM. *CAL* (2024).

[103] Miten Jain, Olsen Hugh E., Benedict Paten, and Mark Akeson. 2016. The Oxford Nanopore MinION: Delivery of Nanopore Sequencing to the Genomics Community. *Genome Biology* (2016).

[104] Miten Jain, Sergey Koren, Karen H Miga, Josh Quick, Arthur C Rand, Thomas A Sasani, John R Tyson, Andrew D Beggs, Alexander T Dilthey, Ian T Fiddes, Sunir Malla, Hannah Marriott, Tom Nieto, Justin O'Grady, Hugh E Olsen, Brent S Pedersen, Arang Rhie, Hollian Richardson, Aaron R Quinlan, Terrance P Snutch, Louise Tee, Benedict Paten, Adam M Phillippy, Jared T Simpson, Nicholas J Loman, and Matthew Loose. 2018. Nanopore Sequencing and Assembly of a Human Genome with Ultra-Long Reads. *Nature Biotechnology* (2018).

[105] JESD209-4B JEDEC. 2017. Low Power Double Data Rate 4 (LPDDR4) Standard.

[106] Sang-Woo Jun, Ming Liu, Sungjin Lee, Jamey Hicks, John Ankcorn, Myron King, Shuotao Xu, and Arvind. 2015. BlueDBM: An Appliance for Big Data Analytics. *ISCA* (2015).

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

[108] Liana E. Kafetzopoulou, Kyriakos Efthymiadis, Kuiama Lewandowski, Ant Crook, Dan Carter, Jane Osborne, Emma Aarons, Roger Hewson, Julian A. Hiscox, Miles W. Carroll, Richard Vipond, and Steven T. Pullan. 2018. Assessment of Metagenomic Nanopore and Illumina Sequencing for Recovering Whole Genome Sequences of Chikungunya and Dengue Viruses Directly from Clinical Samples. *Euro Surveill* (2018).

[109] Dae-Hyun Kang, Min-Soo Kim, Sung-Chul Jeon, Woo-Sung Jung, Jin-Young Park, Gyu-Tae Choo, Dae-Kee Shim, Anil Kavala, Seung-Bum Kim, Kyu-Min Kang, Jae-Hyun Lee, Ki-Young Ko, Hyun-Woo Park, Byung-Joon Min, Chan Yu, Sung-Kyu Yun, Nam Kim, Yong Jung, Seung Seo, Sangwoo Kim, Min-Kyu Lee, Joon-Young Park, Jin-Chul Kim, Young-Su Cha, Kyung Kim, Yong Jo, Hyun Kim, Yong Choi, Jihoon Byun, Jin-Ho Park, Ki Kim, Tae-Hyoung Kwon, Young Min, Chang Yoon, Young Kim, Dong-Hyun Kwak, Eui Lee, Woo-Geun Hahn, Kyu-Seok Kim, Ki Kim, Eui Yoon, Woo-Tae Kim, Inhee Lee, Sang-Ho Moon, Jong Ihm, Do-Seung Byeon, Kyung-Won Song, Sung Hwang, and Chong-Ho Kyung. 2019. A 512Gb 3-Bit/Cell 3D 6th-Generation V-NAND Flash Memory with 82MB/s Write Throughput and 1.2Gb/s Interface. In *ISSCC*.

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

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

[112] Roman Kaplan, Leonid Yavits, and Ran Ginosasr. 2020. BioSEAL: In-Memory Biological Sequence Alignment Accelerator for Large-Scale Genomic Data. In *SYSTOR*.

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

[114] S Karen Khatamifard, Zamshed Chowdhury, Nakul Pande, Meisam Razaviyayn, Chris Kim, and Ulya R Karpuzcu. 2021. GeNVoM: Read Mapping Near Non-Volatile Memory. *TCBB* (2021).

[115] Jeremie S Kim, Damla Senol Cali, Hongyi Xin, Donghyuk Lee, Saugata Ghose, Mohammed Alser, Hasan Hassan, Oguz Ergin, Can Alkan, and Onur Mutlu. 2018. GRIM-Filter: Fast Seed Location Filtering in DNA Read Mapping Using Processing-in-memory Technologies. *BMC Genomics* (2018).

[116] Jeremie S Kim, Minesh Patel, Hasan Hassan, and Onur Mutlu. 2018. Solar-DRAM: Reducing DRAM Access Latency by Exploiting the Variation in Local Bitlines. In *ICCD*.

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

[118] Gunjae Koo, Kiran Kumar Matam, Te I, HV Krishna Giri Narra, Jing Li, Hung-Wei Tseng, Steven Swanson, and Murali Annavaram. 2017. Summarizer: Trading Communication with Computing Near Storage. In *MICRO*.

[119] Sam Kovaka, Yunfan Fan, Bohan Ni, Winston Timp, and Michael C. Schatz. 2021. Targeted Nanopore Sequencing by Real-time Mapping of Raw Electrical Signal with UNCALLED. *Nature Biotechnology* (2021).

[120] Sam Kovaka, Paul W. Hook, Katharine M. Jenike, Vikram Shivakumar, Luke B. Morina, Roham Razaghi, Winston Timp, and Michael C. Schatz. 2024. Uncalled4 Improves Nanopore DNA and RNA Modification Detection via Fast and Accurate Signal Alignment. *Nature Methods* (2024).

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

[122] Andrew H Laszlo, Ian M Derrington, Henry Brinkerhoff, Kyle W Langford, Ian C Nova, Jenny Mae Samson, Joshua J Bartlett, Mikhail Pavlenok, and Jens H Gundlach. 2013. Detection and Mapping of 5-methylcytosine and 5-hydroxymethylcytosine with Nanopore MspA. *PNAS* (2013).

[123] A. H. Laszlo, I. M. Derrington, B. C. Ross, H. Brinkerhoff, A. Adey, I. C. Nova, J. M. Craig, K. W. Langford, J. M. Samson, R. Daza, K. Doering, J. Shendure, and J. H. Gundlach. 2014. Decoding Long Nanopore Sequencing Reads of natural DNA. *Nature Biotechnology* (2014).

[124] Donghyuk Lee, Samira Khan, Lavanya Subramanian, Saugata Ghose, Rachata Ausavarungnirun, Gennady Pekhimenko, Vivek Seshadri, and Onur Mutlu. 2017. Design-Induced Latency Variation in Modern DRAM Chips: Characterization, Analysis, and Latency Reduction Mechanisms. In *SIGMETRICS*.

[125] Joo Hwan Lee, Hui Zhang, Veronica Lagrange, Praveen Krishnamoorthy, Xiaodong Zhao, and Yang Seok Ki. 2020. SmartSSD: FPGA-accelerated Near-Storage Data Analytics on SSD. *CAL* (2020).

[126] Marzieh Lenjani, Patricia Gonzalez, Elaheh Sadredini, Shuangchen Li, Yuan Xie, Ameen Akel, Sean Eilert, Mircea R. Stan, and Kevin Skadron. 2020. Fulcrum: A Simplified Control and Access Mechanism Toward Flexible and Practical In-Situ Accelerators. In *HPCA*.

[127] Marzieh Lenjani and Kevin Skadron. 2022. Supporting Moderate Data Dependency, Position Dependency, and Divergence in PIM-Based Accelerators. *Micro* (2022).

[128] Heng Li. 2018. Minimap2: Pairwise Alignment for Nucleotide Sequences. *Bioinformatics* (2018).

[129] Heng Li and Jue Ruan. 2008. Mapping Short DNA Sequencing Reads and Calling Variants Using Mapping Quality Scores. *Genome research*

(2008).

[130] Luyi Li, Jun Lin, and Zhongfeng Wang. 2021. PipeBSW: A two-stage pipeline structure for Banded Smith-Waterman Algorithm on FPGA. In *ISVLSI*.

[131] Xue-Qi Li, Guang-Ming Tan, and Ning-Hui Sun. 2021. PIM-align: a Processing-In-Memory Architecture for FM-index Search Algorithm. *Journal of Computer Science and Technology* (2021).

[132] Yifan Li and Giulia Guidi. 2024. High-Performance Sorting-Based K-mer Counting in Distributed Memory with Flexible Hybrid Parallelism. In *ICPP*.

[133] Yang Liao, Gordon K Smyth, and Wei Shi. 2013. The Subread Aligner: Fast, Accurate and Scalable Read Mapping by Seed-and-Vote. *Nucleic acids research* (2013).

[134] Sang-Phil Lim, Sang-Won Lee, and Bongki Moon. 2010. FASTer FTL for Enterprise-Class Flash Memory SSDs. In *SNAPI*.

[135] Joël Lindegger, Can Firtina, Nika Mansouri Ghiasi, Mohammad Sadrosadati, Mohammed Alser, and Onur Mutlu. 2024. RawAlign: Accurate, Fast, and Scalable Raw Nanopore Signal Mapping via Combining Seeding and Alignment. *IEEE Access* (2024).

[136] Song Liu, Yi Wang, and Fei Wang. 2016. A Fast Read Alignment Method based on Seed-and-Vote for Next Generation Sequencing. *BMC bioinformatics* (2016).

[137] Yongchao Liu, Douglas L Maskell, and Bertil Schmidt. 2009. CUDASW++: Optimizing Smith-Waterman Sequence Database Searches for CUDA-enabled Graphics Processing Units. *BMC research notes* (2009).

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

[139] Yongchao Liu, Bertil Schmidt, and Douglas L Maskell. 2010. CUDASW++ 2.0: Enhanced Smith-Waterman Protein Database Search on CUDA-enabled GPUs based on SIMT and Virtualized SIMD Abstractions. *BMC research notes* (2010).

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

[141] Kisaru Liyanage, Hasindu Gamaarachchi, Roshan Ragel, and Sri Parameswaran. 2023. Cross Layer Design Using HW/SW Co-Design and HLS to Accelerate Chaining in Genomic Analysis. *TCAD* (2023).

[142] Kisaru Liyanage, Hasindu Gamaarachchi, Hassaan Saadat, Tuo Li, Hiruna Samarakoon, and Sri Parameswaran. 2024. Accelerating Chaining in Genomic Analysis Using RISC-V Custom Instructions. In *DATE*.

[143] Kisaru Liyanage, Hiruna Samarakoon, Sri Parameswaran, and Hasindu Gamaarachchi. 2023. Efficient End-to-End Long-read Sequence Mapping using Minimap2-FPGA integrated with hardware-accelerated Chaining. *Scientific Reports* (2023).

[144] Matthew Loose, Sunir Malla, and Michael Stout. 2016. Real-time Selective Sequencing using Nanopore Technology. *Nat. Methods* (2016).

[145] Qian Lou, Sarath Chandra Janga, and Lei Jiang. 2020. Helix: Algorithm/Architecture co-design for Accelerating Nanopore Genome Base-calling. In *PACT*.

[146] Qian Lou and Lei Jiang. 2018. Brawl: A Spintronics-based Portable Basecalling-In-Memory Architecture for Nanopore Genome Sequencing. *CAL* (2018).

[147] Hang-Ting Lue, Po-Kai Hsu, Ming-Liang Wei, Teng-Hao Yeh, Pei-Ying Du, Wei-Chen Chen, Keh-Chung Wang, and Chih-Yuan Lu. 2019. Optimal Design Methods to Transform 3D NAND Flash into a High-Density, High-Bandwidth and Low-Power Nonvolatile Computing In Memory (nvCIM) Accelerator for Deep-Learning Neural Networks (DNN). In *IEDM*.

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

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

[150] Advait Madhavan, Timothy Sherwood, and Dmitri Strukov. [n. d.]. Race Logic: A hardware acceleration for dynamic programming algorithms. In *ISCA*.

[151] Vikram Sharma Mailthody, Zaid Qureshi, Weixin Liang, Ziyan Feng, Simon Garcia De Gonzalo, Youjie Li, Hubertus Franke, Jinjun Xiong, Jian Huang, and Wen-mei Hwu. 2019. Deepstore: In-storage Acceleration for Intelligent Queries. In *MICRO*.

[152] 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 *ASPLOS*.

[153] Haiyu Mao, Mohammed Alser, Mohammad Sadrosadati, Can Firtina, Akanksha Baranwal, Damla Senol Cali, Aditya Manglik, Nour Almadhoun Alserr, and Onur Mutlu. 2022. GenPIP: In-Memory Acceleration of Genome Analysis via Tight Integration of Basecalling and Read Mapping. In *MICRO*.

[154] Farnood Merrikh-Bayat, Xinjie Guo, Michael Klachko, Mirko Prezioso, Konstantin K Likharev, and Dmitri B Strukov. 2017. High-Performance Mixed-Signal Neurocomputing with Nanoscale Floating-Gate Memory Cell Arrays. *TNNLS* (2017).

[155] Justin Meza, Qiang Wu, Sanjev Kumar, and Onur Mutlu. 2015. A Large-Scale Study of Flash Memory Failures in the Field. In *ACM SIGMETRICS*.

[156] Rino Micheloni, Alessia Marelli, and Kam Eshghi. 2018. Inside Solid State Drives (SSDs). (2018).

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

[158] Karen H. Miga, Sergey Koren, Arang Rhie, Mitchell R. Vollger, Ariel Gershman, Andrey Bzikadze, Shelise Brooks, Edmund Howe, David Porubsky, Glennis A. Logsdon, Valerie A. Schneider, Tamara Potapova, Jonathan Wood, William Chow, Joel Armstrong, Jeanne Fredrickson, Evgenia Pak, Kristof Tigyi, Milinn Kremitzki, Christopher Markovic, Valerie Maduro, Amalia Dutra, Gerard G. Bouffard, Alexander M. Chang, Nancy F. Hansen, Amy B. Wilfert, Françoise Thibaud-Nissen, Anthony D. Schmitt, Jon-Matthew Belton, Siddarth Selvaraj, Megan Y. Dennis, Daniela C. Soto, Ruta Sahasrabudhe, Gulhan Kaya, Josh Quick, Nicholas J. Loman, Nadine Holmes, Matthew Loose, Urvashi Surti, Rosa ana Risques, Tina A. Graves Lindsay, Robert Fulton, Ira Hall, Benedict Paten, Kerstin Howe, Winston Timp, Alice Young, James C. Mullikin, Pavel A. Pevzner, Jennifer L. Gerton, Beth A. Sullivan, Evan E. Eichler, and Adam M. Phillippy. 2020. Telomere-to-telomere Assembly of a Complete Human X Chromosome. *Nature* (2020).

[159] Andrew J Mikalsen and Jaroslaw Zola. 2023. Coriolis: Enabling Metagenomic Classification on Lightweight Mobile Devices. *Bioinform.* (2023).

[160] Onur Mutlu and Can Firtina. 2023. Accelerating Genome Analysis via Algorithm-Architecture Co-Design. In *DAC*.

[161] Gene Myers and Webb Miller. 1995. Chaining multiple-alignment fragments in sub-quadratic time. In *ACM-SIAM Symposium on Discrete Algorithms*.

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

[163] Jisung Park, Roknoddin Azizi, Geraldo F Oliveira, Mohammad Sadrosadati, Rakesh Nadig, David Novo, Juan Gómez-Luna, Myungsuk Kim, and Onur Mutlu. 2022. Flash-Cosmos: In-Flash Bulk Bitwise Operations Using Inherent Computation Capability of NAND Flash Memory. In *MICRO*.

[164] Alexander Payne, Nadine Holmes, Thomas Clarke, Rory Munro, Bisrat J. Debebe, and Matthew Loose. 2021. Readfish Enables Targeted Nanopore Sequencing of gigabase-sized Genomes. *Nature Biotechnology* (2021).

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

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

[167] Minh Pham, Yicheng Tu, and Xiaoyi Lv. 2023. Accelerating BWA-MEM Read Mapping on GPUs. In *ICS*.

[168] Ryan Poplin, Pi-Chuan Chang, David Alexander, Scott Schwartz, Thomas Colthurst, Alexander Ku, Dan Newburger, Jojo Dijamco, Nam Nguyen, Pegah T Afshar, Sam S Gross, Lizzie Dorfman, Cory Y McLean, and Mark A DePristo. 2018. A Universal SNP and Small-Indel Variant Caller Using Deep Neural Networks. *Nature Biotechnology* (2018).

[169] Javier Prado-Martinez, Peter H. Sudmant, Jeffrey M. Kidd, Heng Li, Joanna L. Kelley, Belen Lorente-Galdos, Krishna R. Veeramah, August E. Woerner, Timothy D. O'Connor, Gabriel Santpere, Alexander Cagan, Christoph Theunert, Ferran Casals, Hafid Laayouni, Kasper Munch, Asger Hobolth, Anders E. Halager, Maika Malig, Jessica Hernandez-Rodriguez, Irene Hernando-Herraez, Kay Prüfer, Marc Pybus, Laurel Johnstone, Michael Lachmann, Can Alkan, Dorina Twigg, Natalia Petit, Carl Baker, Fereydoun Hormozdiari, Marcos Fernandez-Callejo, Marc Dabad, Michael L. Wilson, Laurie Stevison, Cristina Camprubí, Tiago Carvalho, Aurora Ruiz-Herrera, Laura Vives, Marta Mele, Teresa Abello, Ivanela Kondova, Ronald E. Bontrop, Anne Pusey, Felix Lankester, John A. Kiyang, Richard A. Bergl, Elizabeth Lonsdorf, Simon Myers, Mario Ventura, Pascal Gagneux, David Comas, Hans Siegismund, Julie Blanc, Lidia Agueda-Calpena, Marta Gut, Lucinda Fulton, Sarah A. Tishkoff, James C. Mullikin, Richard K. Wilson, Ivo G. Gut, Mary Katherine Gonder, Oliver A. Ryder, Beatrice H. Hahn, Arcadi Navarro, Joshua M. Akey, Jaume Bertranpetit, David Reich, Thomas Mailund, Mikkel H. Schierup, Christina Hvilsom, Aida M. Andrés, Jeffrey D. Wall, Carlos D. Bustamante, Michael F. Hammer, Evan E. Eichler, and Tomas Marques-Bonet. 2013. Great Ape Genetic Diversity and Population History. *Nature* (2013).

[170] Arthur C Rand, Miten Jain, Jordan M Eizenga, Audrey Musselman-Brown, Hugh E Olsen, Mark Akeson, and Benedict Paten. 2017. Mapping DNA Methylation with High-throughput Nanopore Sequencing. *Nature Methods* (2017).

[171] Arang Rhie, Sergey Nurk, Monika Cechova, Savannah J. Hoyt, Dylan J. Taylor, Nicolas Altemose, Paul W. Hook, Sergey Koren, Mikko Rautiainen, Ivan A. Alexandrov, Jamie Allen, Mobin Asri, Andrey V. Bzikadze, Nae-Chyun Chen, Chen-Shan Chin, Mark Diekhans, Paul Flicek, Giulio Formenti, Arkarachai Fungtammasan, Carlos Garcia Giron, Erik Garrison, Ariel Gershman, Jennifer L. Gerton, Patrick G.S. Grady, Andrea Guarracino, Leanne Haggerty, Reza Halabian, Nancy F. Hansen, Robert Harris, Gabrielle A. Hartley, William T. Harvey, Marina Haukness, Jakob Heinz, Thibaut Hourlier, Robert M. Hubley, Sarah E. Hunt, Stephen Hwang, Miten Jain, Rupesh K. Kesharwani, Alexandra P. Lewis, Heng Li, Glennis A. Logsdon, Julian K. Lucas, Wojciech Makalowski, Christopher Markovic, Fergal J. Martin, Ann M. Mc Cartney, Rajiv C. McCoy, Jennifer McDaniel, Brandy M. McNulty, Paul Medvedev, Alla Mikheenko, Katherine M. Munson, Terence D. Murphy, Hugh E. Olsen, Nathan D. Olson, Luis F. Paulin, David Porubsky, Tamara Potapova, Fedor Ryabov, Steven L. Salzberg, Michael E.G. Sauria, Fritz J. Sedlazeck, Kishwar Shafin, Valery A. Shepelev, Alaina Shumate, Jessica M. Storer, Likhitha Surapaneni, Angela M. Taravella Oill, Françoise Thibaud-Nissen, Winston Timp, Marta Tomaszkiewicz, Mitchell R. Vollger, Brian P. Walenz, Allison C. Watwood, Matthias H. Weissensteiner, Aaron M. Wenger, Melissa A. Wilson, Samantha Zarate, Yiming Zhu, Justin M. Zook, Evan E. Eichler, Rachel J. O'Neill, Michael C. Schatz, Karen H. Miga, Kateryna D. Makova, and Adam M. Phillippy. 2023. The Complete Sequence of a Human Y Chromosome. *Nature* (2023).

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

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

[174] Guillaume Rizk and Dominique Lavenier. 2010. GASSST: Global Alignment Short Sequence Search Tool. *Bioinformatics* (2010).

[175] J Romiguier, Philippe Gayral, Marion Ballenghien, Arnaud Bernard, Vincent Cahais, A Chenuil, Ylenia Chiari, R Dernat, L Duret, Nicolas Faivre, et al. 2014. Comparative Population Genomics in Animals Uncovers the Determinants of Genetic Diversity. *Nature* (2014).

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

[177] Harisankar Sadasivan, Milos Maric, Eric Dawson, Vishanth Iyer, Johnny Israeli, and Satish Narayanasamy. 2023. Accelerating Minimap2 for Accurate Long Read Alignment on GPUs. *Journal of biotechnology and biomedicine* (2023).

[178] Harisankar Sadasivan, Daniel Stiffler, Ajay Tirumala, Johnny Israeli, and Satish Narayanasamy. 2023. Accelerated Dynamic Time Warping on GPU for Selective Nanopore Sequencing. *J. Biomed Biotechnol* (2023).

[179] Harisankar Sadasivan, Jack Wadden, Kush Goliya, Piyush Ranjan, Robert P. Dickson, David Blaauw, Reetuparna Das, and Satish Narayanasamy. 2023. Rapid Real-time Squiggle Classification for Read Until Using RawMap. *Arch. Clin. Biomed. Res.* (2023).

[180] Suneth Samarasinghe, Pubudu Premathilaka, Wishma Herath, Hasindu Gamaarachchi, and Roshan Ragel. 2021. Energy Efficient Adaptive Banded Event Alignment using OpenCL on FPGAs. In *ICIAfS*.

[181] Nikola Samardzic, Weikang Qiao, Vaibhav Aggarwal, Mau-Chung Frank Chang, and Jason Cong. 2020. Bonsai: High-Performance Adaptive Merge Tree Sorting. In *ISCA*.

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

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

[184] Doruk Sart, Abdullah Mueen, Walid Najjar, Eamonn Keogh, and Vit Niennattrakul. 2010. Accelerating Dynamic Time Warping Subsequence Search with GPUs and FPGAs. In *ICDM*.

[185] Damla Senol Cali, Gupreet Kalsi, Zulal Bingöl, Lavanya Subramanian, Can Firtina, Jeremie Kim, Rachata Ausavarungnirun, Mohammed Alser, Anant Nori, Juan Luna, et al. 2020. GenASM: A High-Performance, Low-Power Approximate String Matching Acceleration Framework for Genome Sequence Analysis. In *MICRO*.

[186] Damla Senol Cali, Jeremie S Kim, Saugata Ghose, Can Alkan, and Onur Mutlu. 2018. Nanopore Sequencing Technology and Tools for Genome Assembly: Computational Analysis of the Current State, Bottlenecks and Future Directions. *Briefings in Bioinformatics* (2018).

[187] Sudharsan Seshadri, Mark Gahagan, Sundaram Bhaskaran, Trevor Bunker, Arup De, Yanqin Jin, Yang Liu, and Steven Swanson. 2014. Willow: A User-Programmable SSD. In *USENIX OSDI*.

[188] Taha Shahroodi, Gagandeep Singh, Mahdi Zahedi, Haiyu Mao, Joel Lindegger, Can Firtina, Stephan Wong, Onur Mutlu, and Said Hamdioui. 2023. Swordfish: A Framework for Evaluating Deep Neural Network-based Basecalling using Computation-In-Memory with Non-Ideal Memristors. In *MICRO*.

[189] Jay Shendure, Shankar Balasubramanian, George M. Church, Walter Gilbert, Jane Rogers, Jeffery A. Schloss, and Robert H. Waterston. 2017. DNA Sequencing at 40: Past, Present and Future. *Nature* (2017).

[190] Po Jui Shih, Hassaan Saadat, Sri Parameswaran, and Hasindu Gamaarachchi. 2023. Efficient Real-time Selective Genome Sequencing on Resource-Constrained Devices. *GigaScience* (2023).

[191] Wonbo Shim and Shimeng Yu. 2022. GP3D: 3D NAND Based In-Memory Graph Processing Accelerator. *JETCAS* (2022).

[192] Jung-Yeon Shin, Zhi-Lin Xia, Ning-Yi Xu, Rui Gao, Xiao-Fei Cai, Seungryoul Maeng, and Fang-Hsien Hsu. 2009. FTL Design Exploration in Reconfigurable High-Performance SSD for Server Applications. In *ICS*.

[193] Vikram S. Shivakumar, Omar Y. Ahmed, Sam Kovaka, Mohsen Zakeri, and Ben Langmead. 2024. Sigmoni: Classification of Nanopore Signal with a Compressed Pangenome Index. *Bioinfromatics* (2024).

[194] Jared T Simpson, Rachael E Workman, P C Zuzarte, Matei David, L J Dursi, and Winston Timp. 2017. Detecting DNA Cytosine Methylation using Nanopore Sequencing. *Nature Methods* (2017).

[195] Gagandeep Singh, Mohammed Alser, Kristof Denolf, Can Firtina, Alireza Khodamoradi, Meryem Banu Cavlak, Henk Corporaal, and Onur Mutlu. 2024. RUBICON: A Framework for Designing Efficient Deep Learning-Based Genomic Basecallers. *Genome Biology* (2024).

[196] Gagandeep Singh, Mohammed Alser, Alireza Khodamoradi, Kristof Denolf, Can Firtina, Meryem Banu Cavlak, Henk Corporaal, and Onur Mutlu. 2024. RUBICON: A Framework for Designing Efficient Deep Learning-Based Genomic Basecallers. *Genome Biology* (2024).

[197] Wei Song, Dirk Koch, Mikel Luján, and Jim Garside. 2016. Parallel Hardware Merge Sorter. In *FCCM*.

[198] William Stephenson, Roham Razaghi, Steven Busan, Kevin M. Weeks, Winston Timp, and Peter Smibert. 2022. Direct Detection of RNA Modifications and Structure using Single-Molecule Nanopore Sequencing. *Cell Genomics* (2022).

[199] Aaron Stillmaker and Bevan Baas. 2017. Scaling Equations for the Accurate Prediction of CMOS Device Performance from 180nm to 7nm. *Integration* (2017).

[200] VK Sundaresan, Sanjay Nichani, N Ranganathan, and Ravi Sankar. 1992. A VLSI Hardware Accelerator for Dynamic Time Warping. In *ICPR*.

[201] Nathaly M Sweeney, Shareef A Nahas, Shimul Chowdhury, Sergey Batalov, Michelle Clark, Sara Caylor, Julie Cakici, John J Nigro, Yan Ding, Narayanan Veeraraghavan, et al. 2021. Rapid Whole Genome Sequencing Impacts Care and Resource Utilization in Infants with Congenital Heart Disease. *NPJ Genomic Medicine* (2021).

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

[203] Shuhei Tanakamaru, Yuki Yanagihara, and Ken Takeuchi. 2013. Error-Prediction LDPC and Error-Recovery Schemes for Highly Reliable Solid-State Drives (SSDs). *IEEE J. Solid-State Circuits* (2013).

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

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

[206] Po-Hao Tseng, Feng-Ming Lee, Yu-Hsuan Lin, Liang-Yu Chen, Yung-Chun Li, Han-Wen Hu, Yun-Yuan Wang, Chih-Chang Hsieh, Ming-Hsiu Lee, Hsiang-Lan Lung, et al. 2020. In-Memory-Searching Architecture Based on 3D-NAND Technology with Ultra-High Parallelism. In *IEDM*.

[207] Yatish Turakhia, Gill Bejerano, and William J Dally. 2018. Darwin: A Genomics Co-processor Provides up to 15,000 x Acceleration on Long Read Assembly. In *ASPLOS*.

[208] UMC. [n. d.]. 55 / 65 / 90nm, https://www.umc.com/en/Product/technologies/Detail/55_65_90nm.

[209] Hasitha Muthumala Waidyasooriya and Masanori Hariyama. 2015. Hardware-Acceleration of Short-Read Alignment based on the Burrows-Wheeler Transform. *TPDS* (2015).

[210] Yuk Kei Wan, Christopher Hendra, Ploy N Pratanwanich, and Jonathan Göke. 2022. Beyond Sequencing: Machine Learning Algorithms Extract Biology Hidden in Nanopore Signal Data. *Trends in Genetics* (2022).

[211] Panni Wang, Feng Xu, Bo Wang, Bin Gao, Huaqiang Wu, He Qian, and Shimeng Yu. 2018. Three-Dimensional NAND flash for Vector–Matrix Multiplication. *TVLSI* (2018).

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

[213] Yitu Wang, Shiyu Li, Qilin Zheng, Linghao Song, Zongwang Li, Andrew Chang, and Yiran Chen. 2024. NDSEARCH: Accelerating Graph-Traversal-Based Approximate Nearest Neighbor Search through Near Data Processing. In *ISCA*.

[214] Yuyue Wang, Xiurui Pan, Yuda An, Jie Zhang, and Glenn Reinman. 2024. BeaconGNN: Large-Scale GNN Acceleration with Out-of-Order Streaming In-Storage Computing. In *HPCA*.

[215] Yunhao Wang, Yue Zhao, Audrey Bollas, Yuru Wang, and Kin Fai Au. 2021. Nanopore Sequencing Technology, Bioinformatics and Applications. *Nature Biotechnology* (2021).

[216] Shan Wei, Zachary R Weiss, and Zev Williams. 2018. Rapid Multiplex Small DNA Sequencing on the MinION Nanopore Sequencing Platform. *G3 Genes|Genomes|Genetics* (2018).

[217] R. R. Wick, L. M. Judd, and K. E. Holt. 2019. Performance of Neural Network Basecalling Tools for Oxford Nanopore Sequencing. *Genome Biology* (2019).

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

[219] Richard Wilton, Tamas Budavari, Ben Langmead, Sarah J Wheelan, Steven L Salzberg, and Alexander S Szalay. 2015. Arioc: High-Throughput Read Alignment with GPU-accelerated Exploration of the Seed-and-Extend Search Space. *PeerJ* (2015).

[220] Rachael E. Workman, Alison D. Tang, Paul S. Tang, Miten Jain, John R. Tyson, Roham Razaghi, Philip C. Zuzarte, Timothy Gilpatrick, Alexander Payne, Joshua Quick, Norah Sadowski, Nadine Holmes, Jaqueline Goes de Jesus, Karen L. Jones, Cameron M. Soulette, Terrance P. Snutch, Nicholas Loman, Benedict Paten, Matthew Loose, Jared T. Simpson, Hugh E. Olsen, Angela N. Brooks, Mark Akeson, and Winston Timp. 2019. Nanopore Native RNA Sequencing of a Human Poly(A) Transcriptome. *Nature Methods* (2019).

[221] Lisa Wu, David Bruns-Smith, Frank A Nothaft, Qijing Huang, Sagar Karandikar, Johnny Le, Andrew Lin, Howard Mao, Brendan Sweeney, Krste Asanovic, et al. 2019. FPGA Accelerated INDEL Realignment in the Cloud. In *HPCA*.

[222] Z. Wu, K. Hammad, A. Beyene, Y. Dawji, E. Ghafar-Zadeh, and S. Magierowski. 2022. An FPGA Implementation of a Portable DNA Sequencing Device Based on RISC-V. In *IEEE International New Circuits*

and Systems Conference (NEWCAS).

[223] Z. Wu, K. Hammad, E. Ghafar-Zadeh, and S. Magierowski. 2020. FPGA-Accelerated 3rd Generation DNA Sequencing. *IEEE Transactions on Biomedical Circuits and Systems (TBCS)* (2020).

[224] Xilinx 2021. *SmartSSD Computational Storage Drive Installation and User Guide.* Xilinx. https://fpga.eetrend.com/files/2022-02/wen_zhang_/100558024-244024-ug1382-smartssd-csd.pdf

[225] Heliang Xin, Jason Greth, Justin Emmons, Gennady Pekhimenko, Carl Kingsford, Can Alkan, and Onur Mutlu. 2015. Shifted Hamming distance: a fast and accurate SIMD-friendly filter to accelerate alignment verification in read mapping. *Bioinformatics* (2015).

[226] Hongyi Xin, David Lee, Farhad Hormozdiari, Shanshan Yedkar, Onur Mutlu, and Can Alkan. 2013. Accelerating Read Mapping with FastHASH. *BMC Genomics* (2013).

[227] Heliang Xin, Shihe Nahar, Rui Zhu, Justin Emmons, Gennady Pekhimenko, Carl Kingsford, Can Alkan, and Onur Mutlu. 2016. Optimal seed solver: optimizing seed selection in read mapping. *Bioinformatics* (2016).

[228] Z. Xu, Y. Mai, D. Liu, W. He, X. Lin, C. Xu, L. Zhang, X. Meng, J. Mafofo, W.A. Zaher, et al. 2021. Fast-bonito: A Faster Deep Learning Based Basecaller for Nanopore Sequencing. *Artificial Intelligence in the Life Sciences* (2021).

[229] Ramesh Yelagandula, Aleksandr Bykov, Alexander Vogt, Robert Heinen, Ezgi Özkan, Marcus Martin Strobl, Juliane Christina Baar, Kristina Uzunova, Bence Hajdusits, Darja Kordic, et al. 2021. Multiplexed Detection of SARS-CoV-2 and Other Respiratory Infections in High Throughput by SARSeq. *Nature Communications* (2021).

[230] Jia Zeng, Hao Cai, Hao Peng, Haotian Wang, Yadong Zhang, and Tatsuya Akutsu. 2020. Causalcall: Nanopore Basecalling using a Temporal Convolutional Network. *Frontiers in Genetics* (2020).

[231] Alberto Zeni, Giulia Guidi, Marquita Ellis, Nan Ding, Marco D Santambrogio, Steven Hofmeyr, Aydin Buluc, Leonid Oliker, and Katherine Yelick. 2020. Logan: High Performance GPU-based X-drop Long-Read Alignment. In *IPDPS*.

[232] F. Zhang, S. Angizi, N. A. Fahmi, W. Zhang, and D. Fan. 2021. PIM-Quantifier: A Processing-In-Memory Platform for mRNA Quantification. In *DAC*.

[233] Fan Zhang, Shaahin Angizi, Jiao Sun, Wei Zhang, and Deliang Fan. 2023. Aligner-D: Leveraging In-DRAM Computing to Accelerate DNA Short Read Alignment. *JETCAS* (2023).

[234] Haowen Zhang, Haoran Li, Chirag Jain, Haoyu Cheng, Kin Fai Au, Heng Li, and Srinivas Aluru. 2021. Real-time Mapping of Nanopore Raw Signals. *Bioinformatics* (2021).

[235] Kai Zhao, Wenzhe Zhao, Hongbin Sun, Xiaodong Zhang, Nanning Zheng, and Tong Zhang. 2013. LDPC-in-SSD: Making Advanced Error Correction Codes Work Effectively in Solid State Drives. In *FAST 13*.

[236] Hongtao Zhong, Zhonghao Chen, Wenqin Huangfu, Chen Wang, Yixin Xu, Tianyi Wang, Yao Yu, Yongpan Liu, Vijaykrishnan Narayanan, Huazhong Yang, et al. 2023. ASMCap: An Approximate String Matching Accelerator for Genome Sequence Analysis Based on Capacitive Content Addressable Memory. *DAC* (2023).

[237] You Zhou, Fei Wu, Ping Huang, Xubin He, Changsheng Xie, and Jian Zhou. 2015. An Efficient Page-level FTL to Optimize Address Translation in Flash Memory. In *EuroSys*.

[238] Farzaneh Zokaee, Hamid R Zarandi, and Lei Jiang. 2018. Aligner: A Process-In-Memory Architecture for Short Read Alignment in ReRAMs. *CAL* (2018).

[239] F. Zokaee, M. Zhang, and L. Jiang. 2019. FindeR: Accelerating FM-index-based Exact Pattern Matching in Genomic Sequences through ReRAM Technology. In *PACT*.

[240] Chen Zou and Andrew A Chien. 2022. Assasin: Architecture Support for Stream Computing to Accelerate Computational Storage. In *MICRO*.